Traducción de juegos LUDI

Todos los mensajes mostrados en LUDI están definidos en formato json reconocible por Google Translator Toolkit (GTT).

GTT: https://translate.google.com/toolkit

XX representa el idioma. Actualmente XX puede ser ES, EN y EO (esperanto).

Los distintos json traducibles son:
/client/data/kernel/kernelXX.json
/client/data/lib/libMessagesXX.json
/client/data/games/<gameName>/gameMessagesXX.json

Ejemplos para kernelXX.json:

```
   "You mark o1 for 2-object actions": {
      "message": "Marcas %o1 para las acciones con dos objetos"
   },
   "nowhere": {
      "message": "a ningún sitio"
   },
   "Actions on o1": {
      "message": "Acciones sobre %o1"
   },
```

Ejemplo para libMessagesXX.json y gameMessagesXX.json

```
   "items.vagabunda.desc": {
      "message": "Es una sucia vagabunda. Está embarazada y está vestida con una capa mojada."
   },
   "items.vagabunda.txt": {
      "message": "vagabunda"
   },
  "messages.DLG_no_da_o1_porque_falta_o2.txt": {
   "message": "Si quieres %o1 deberías traerme %o2."
 },
```

Explicación:
“%” + número indica que el mensaje recibe un parámetro que debe ser reemplazado. En general se refieren a items (objetos) del juego %o

Los mensajes item.<ID>.txt son el texto corto que se muestra cuando se menciona un objeto. Es lo que se expande en los parámetros %o.

Los mensajes item.<ID>.desc son la descripción de un item, ya sea localidad, objeto, PNJ o PJ. También puedes renunciar a usar esta entrada y definir una función desc() asociada al item dentro de game_reactions.js

Traducción usando GTT.

Con GTT subes el fichero json que deseas traducir, indicando el idioma de origen y el idioma o idiomas destino. GTT creará una traducción automática de los mismos.

A partir de ahí tienes dos opciones, o te descargas los ficheros traducidos y los editar manualmente fuera de GTT, o bien, realizar la traducción total en GTT (permite traducción colaborativa dando permisos a otros) y descargarla cuando acabes.

El GTT muchas veces trastoca los parámetros %, por lo que es necesario una revisión manual de los mismos.

Una posible manera de actuar es la siguiente:

1. Exportar inmediatamente los ficheros json generados, poniéndolo en el nombre de fichero adecuado, con dos letras del idioma adecuado;
2. Reparar los parámetros %. A veces basta con buscar/reemplazar ciertas cadenas.
3. Con el paso 2 acabado, ya se podría modificar el fichero about.json del juego para incorporar el nuevo idioma entre los elegibles para jugar, aunque sea en modo “indio apache”.
4. Quedaría sólo finalizar el remate del fichero de idioma. Como ya existe uno o más idiomas, esas otras versiones pueden ayudar en la traducción.

Ficheros auxiliares con información de cada ítem dependiente del idioma.

Los siguientes ficheros auxiliares se usan para aportar información idiomática sobre cada item del juego:

/client/data/lib/libMessagesByLanXX.json
/client/data/games/<gameName>/gameMessagesByLanXX.json

Ejemplo:

```
"items.cazador.articulo": {
    "message": "el"
},
```

Se trataría (en desarrollo) de aportar cosas como el género, número, persona de un item, y si es nombre propio o no. Para verbos, para dar información sobre cómo conjugarlos, etc.

Sobre la generación del fichero de mensajes de un juego

Como se ha dicho antes, en el fichero gameMessagesES.json (suponiendo idioma español) se definen los nombres a mostrar y descripciones de los ítems, usando las entradas "items.<id_item>.txt" y "items.<id_item>.desc".

Para las frases mostradas en un texto se utilizan las entradas "messages.<id_mensaje>.txt".

En primera instancia, en LUDI, se puede mostrar un texto con la instrucción CA_ShowMsg (mensajeId, parametros). Ejemplo:

CA_ShowMsg (“No tienes con qué coger %o1”, [itemId]);

Donde itemId es un identificador de un objeto del juego. Supongamos que apunta a “la pelota”.

Cuando se ejecute la instrucción, se buscará si existe en gameMessagesES.json el texto con identificador “mensajes.No tienes con qué coger %o1.txt”. Suponiendo que no lo hemos creado, se mostrará el texto directamente (aunque realizando la sustitución del parámetro %o1) aunque subrayado para destacar que no está en el fichero de mensajes.

Esto permite un desarrollo rápido sin tener que estar escribiendo cada frase que usemos en el juego en gameMessagesES.json. Sin embargo, pueden haber frases muy largas y no es muy eficiente usar las propias frases como identificadores. Además, el juego final no debería mostrar frases subrayadas, sino que todas estén definidas, para permitir la traducción del juego a otros idiomas.

La manera de conseguirlo es usando declaraciones de identificadores de texto dentro del juego, con la instrucción GD_CreateMsg (languageIndex, messageId, text). Para el ejemplo anterior, la cosa quedaría:

```
GD_CreateMsg (1, "te_falta_o1", "No tienes con qué coger %o1");
CA_ShowMsg("te_falta_o1", [itemId]);
```

Ahora la cosa va mucho mejor, el texto mostrado no se mostrará subrayado (aunque no esté definido aún en el fichero gameMessagesES.json) y facilita, cuando hayas acabado el juego, usar las llamadas a GD_CreateMsg para generar las líneas que se deberán añadir al fichero gameMessagesES.json.

Por ahora, la forma de conseguirlo es con un procedimiento semiautomático:

1. Extraer las líneas con GD_CreateMsg de game_reactions.js
2. Poner el navegador en modo desarrollo (depende de cada navegador) y pausar la ejecución justo después de cargar los datos del juego.
3. Definir ludi_root.GD_CreateMsg2JSON = true
4. Ejecutar las líneas extraídas en el paso 1 desde la consola de desarrollo
5. Copiar el texto generado y meterlo en gameMessagesES.json

Puede parecer un poco farragoso, pero sólo una vez: cuando ya tienes el juego acabado y quieres que pueda ser traducido a otros idiomas.

Usar la función GD_CreateMsg también facilita la legibilidad del código, ya que puedes agrupar los mensajes usados en una sección de código al principio de la misma, con lo que

el código en vez de estar plagado de texto a mostrar, sólo mostrará los identificadores de texto que indican el mensaje que deseas mostrar.

Funciones para visualizar texto

Además de **CA_ShowMsg**, hay un visualizador de diálogos, funciones **CA_QuoteBegin** (itemId, messageid, param, flagContinues) y **CA_QuoteContinues** (messageid, param, flagContinues).

Estas funciones permiten formatear la salida de texto cuando “habla” un personaje. Mostramos un ejemplo complejo, usando un identificadores de texto dinámico, que señala al texto a mostrar según sea con quién se habla:

```
GD_CreateMsg (1, "DLG_vagabunda_recibe_arma_1", "¡Muchas gracias, caballero!");
GD_CreateMsg (1, "DLG_cazador_recibe_arma_1", "¡Muchas gracias, guapa!");
GD_CreateMsg (1, "DLG_recibe_arma_2", "Creo que me puede ser útil. Hasta ahora le veía con desconfianza pero creo que me puedo fiar de ti. No sé por qué, pero me resultas familiar.");

CA_QuoteBegin (par_c.item2Id, "DLG_" + par_c.item2Id + "_recibe_arma_1", [], false);
CA_QuoteContinues ("DLG_recibe_arma_2");
```

Explicación:

CA_QuoteBegin si no acaba con el parámetro flagContinues a false, genera una salida de texto de tipo **Fulanito**: «¿Qué quieres, forastero?», abriendo y cerrando la cita.

Pero si lo que va a decir necesita ser generado con varias instrucciones, haces que termine con false (sólo se abre la cita, pero no se cierra), y vas usando repetidamente la función CA_QuoteContinues con flagContinues a false, hasta una última CA_QuoteContinues que no acabe con false: esto generará el fin de la cita.