Grado de Ingeniería Informática

TRABAJO DE FIN DE GRADO

Modernización de un videojuego y sus herramientas de trabajo

David Nava Fernandez

Dirigido por Marc Sànchez-Artigas

UNIVERSITAT ROVIRA I VIRGILI

Tarragona
2022

Agradecimientos

Este proyecto no sería producto de una experiencia única de no haber sido por los compañeros de 4Legacy, los cuales han confiado en mí y me han proporcionado el código, el conocimiento y la oportunidad de colaborar en un proceso tan duradero e importante para todos ellos.

Por otra parte, me gustaría agradecer a todas las personas de mi entorno que han conseguido convivir con mi estrés y la difícil gestión de mis sentimientos, sobre todo en aquellas épocas en las cuales el proyecto se ha visto en situación crítica.

Finalmente, quisiera dar las gracias a Marc Sàncez-Artigas, quien ha guiado y gestionado el proceso de entrega de este trabajo, ofreciendo su ayuda y disponibilidad a lo largo del transcurso de este.

Resumen

Colaboración con un equipo de tres jóvenes en el estudio e investigación del código de un videojuego existente con más de 5 millones de líneas de código y 8 mil archivos.

El objetivo principal es la actualización y modernización del juego tras el diseño y la implementación de nuevas características. Para realizar dicha tarea se ha necesitado de la creación de herramientas que permiten modificar datos ya existentes por parte de usuarios con desconocimiento de programación.

La meta supone la preparación de un entorno donde se permite realizar un mantenimiento eficiente y continuo al videojuego una vez lanzado al mercado.

El resultado final ha conllevado en el lanzamiento oficial del videojuego de forma pública tras un proceso de más de dos años y una pequeña campaña de marketing.

Índice

Índice de figuras

1. Introducción

Inicialmente, se puede encontrar una breve introducción al proyecto, el cual he destinado mi trabajo final de investigación. En esta introducción se encuentra descrito de forma breve el videojuego cuyo desarrollo he sido partícipe, así como el origen y las tareas a realizar en este.

Este proyecto se basa en la experiencia de colaborar en la modernización de un videojuego ya existente.

Dicho proyecto se ha decidido llamar 4Legacy, y conjuntamente con otros 3 compañeros (quienes, más adelante, se encuentran detalladas sus principales funciones), se lleva a cabo con el fin de poder crear y disponer de un entorno propio en el cual, se consigue compilar el código, utilizar una base de datos, configurar sus datos maestros y realizar modificaciones de diseño.

4Legacy surge de un videojuego cuyo título se lanzó con el nombre 4Story en el 2008, años más tarde, el código de algunas de sus versiones quedó publicado en internet. Para la creación de nuestra propia versión de 4Story, partimos del uso de uno de estos códigos públicos, concretamente, el correspondiente a la versión 3.5.

El código publicado sufre algunas limitaciones, tales como la no existencia de documentación o la corrupción de los comentarios de código, la dificultad en crear el entorno de compilación, la pérdida de información y tablas de la base de datos, así como de algunos de sus archivos con datos maestros. Por otra parte, la manipulación de dicho código por parte de la comunidad ha desencadenado en algunos exploits que resultan ser importantes de solucionar puesto, que muchos son conocidos de entre los programadores más destacados y experimentados del sector 4Story y es el primer camino a seguir a la hora de intentar hacer un uso malintencionado de nuestro videojuego el día que decide lanzarse. Por último, una de las tareas a realizar, la cual cubre gran parte de mi labor en 4Legacy, es la necesidad de diseñar y proporcionar herramientas que permiten la manipulación de datos de forma sencilla, eficiente y, pensadas para usuarios con desconocimiento de programación (ya que algunos de mis compañeros no disponen de experiencia en programación y centran su labor en la manipulación e investigación de estos datos).

La finalidad de este proyecto es conseguir llegar a una solución estable y preparada para realizar un despliegue del videojuego de forma pública, aportando a los usuarios ciertas experiencias únicas propias de nuestro servidor y de gran agrado para la comunidad de jugadores de 4Story.

Conseguí entrar en dicho proyecto, ya que conocía (de forma digital) a uno de los creadores de este, y tras haberme interesado por la idea y haber conocido su objetivo, decidí proponer mi ayuda en todo lo posible. Mi punto de partida ha sido aquel en el cual el entorno de compilación ya se encontraba creado, de forma que mi primera misión era comenzar a estudiar y entender lo imprescindible, además de todo aquello que afectaba al código más destacado en el cual se desea interactuar.

Han transcurrido aproximadamente tres años desde el inicio de este proyecto (dos y medio para mí), en los cuales se comprenderá más adelante en la documentación las diferentes ocupaciones que yo he realizado.

2. Descripción general

Para comprender con más detalle el funcionamiento y la forma de trabajar en 4Legacy, se describe en este apartado de forma detallada información sobre los miembros del equipo. También es necesario conocer qué es y cómo funciona 4Story, para así comprender la necesidad de cada una de las tareas que he realizado en el proyecto.

2.1. Equipo

Durante la mayor parte del proyecto, hemos sido cuatro los integrantes del equipo, las funciones de las cuales son:

- Marc: Fundador y líder del grupo. Su tarea principal es dirigir el proyecto, siendo también el representante de cara a la comunidad. Ha aportado el capital necesario en cuanto a las inversiones requeridas. Es de gran importancia también el vínculo de contactos que ha creado para dar soporte al proyecto a pequeña escala. Es el más flexible de todos, ya que ha dedicado la mayor parte de su tiempo a aprender sobre todos los ámbitos para así poder ayudar en las tareas que más atrasados quedamos. Durante los dos primeros años sus tareas han sido similares a las de Enrique, destinando gran parte de su tiempo a manipular la base de datos y los archivos con datos maestros. El último año lo ha dedicado a aprender y realizar cursos de programación para poder hacerse cargo del mantenimiento del código.
- Enrique: Ha conseguido ser uno de los expertos en manipulación de datos de 4Story más importantes de la actualidad, ocupando prácticamente todo su tiempo en investigar, manipular y probar datos almacenados tanto en base de datos como en los archivos mencionados anteriormente. Proporciona soporte también a la hora de crear nuevos diseños e ideas para valorar la compatibilidad de dichos datos con los ya existentes.
- Raúl: Programador experimentado cuya labor es principalmente la de trabajar en el motor gráfico del juego. Ha conseguido adaptar 4Legacy a Direct 10 (al ser lanzado en 2008 este rasgo no se encontraba implementado). Anteriormente a mi llegada y durante mi periodo de aprendizaje también estaba a cargo del mantenimiento del código. En cuestiones de seguridad, Raúl es quien más soluciones ha aportado hasta ahora.
- **David:** Mi labor principal ha sido realizar el diseño y la creación de aplicaciones con interfaz gráfica que permiten cargar y manipular datos del juego. Dichas aplicaciones deben de estar orientadas a usuarios con desconocimiento de programación (como por ejemplo Enrique). Aparte, he sido mayoritariamente el encargado del mantenimiento del código, así como el desarrollador de nuevas características de este. Otro de mis objetivos ha sido la implementación de un launcher el cual fundamentalmente permite a los usuarios actualizar el juego y conectarse.

2.1. 4Story

2.1.1. ¿Qué es?

4Story es un videojuego de tipo MMORPG (massively multiplayer online role-playing game) en el cual el usuario dispone de un personaje principal y consigue mediante el cumplimiento de misiones y la destrucción de criaturas enemigas alcanzar el nivel 90. Durante este procedimiento es libre de encontrarse con otros jugadores del bando contrario y pelear contra ellos.

Una vez alcanzado el nivel máximo, el siguiente objetivo acostumbra a ser disfrutar de la experiencia de pelear continuamente contra otros jugadores mediante modos de juego y batallas a campo abierto. Las recompensas suelen proporcionar mejoras al personaje, las cuales también pueden ser obtenidas mediante la destrucción de criaturas.

De forma mensual, se producen torneos para proclamar el mejor luchador de cada una de las clases que este juego distingue (guerrero, mago, sacerdote, conjurador, arquero y corredor de las sombras) y suelen venir acompañados de gran cantidad de jugadores en forma de espectadores.

Figura 1. Muestra visual del juego

2.1.1. ¿Cómo funciona?

Para llevar esto a cabo, 4Story está organizado según la común arquitectura cliente-servidor. En este modelo encontramos un cliente, el cual se encarga generalmente de todos los aspectos visuales del juego, mientras que el servidor es el encargado de controlar el comportamiento del juego y realizar la comunicación entre diferentes usuarios, así como con la base de datos.

El **cliente** es el responsable en una primera instancia de cargar todos los componentes visuales del juego (como por ejemplo los diferentes frames), a su misma vez, cumple con la necesidad de cargar los diferentes archivos situados en la carpeta principal del juego y que contienen información generalmente de carácter gráfico.

Estos archivos son los que permiten manipular las aplicaciones que comentaremos más adelante.

Una vez el cliente ha completado la carga de todos los componentes, permite al usuario visualizar el menú de login. En dicho menú, el usuario accede con sus credenciales a su cuenta (realizando llamadas de comprobación al servidor) y termina seleccionando el personaje con el cual desea iniciar la sesión.

Durante el resto del tiempo, el cliente centra su comportamiento en realizar el render de todos los componentes visuales que el personaje visualiza, realizando movimientos, actualizaciones o efectos visuales.

El **servidor,** por otra parte, está organizado en forma de servicios, permitiendo así ser ejecutado como una aplicación corriente y dando la posibilidad de localizar estos servicios en un host remoto (el cual será necesario a la hora de realizar el lanzamiento del juego para soportar grandes cargas de jugadores y encontrarse disponible 24 horas).

A lo largo de esta documentación se detalla y sobre el uso de los dos servicios principales, los cuales reciben el nombre de TMapSvr y TWorldSvr. Entre la utilidad y las funcionalidades de ambos deberían de existir evidentes distinciones, pero el hecho de no disponer de una documentación pública y haber sufrido la manipulación del código por parte de diferentes usuarios hace que en muchos casos ambos sean flexibles y cumplan funciones similares.

Por lo general, el TMap es el encargado de resolver todas las solicitudes corrientes del juego en un estado normal, es decir, que no necesitan interactuar ni informar a otros servicios. Por otro lado, el TWorld es el encargado de interactuar con todas las peticiones y/o sub-peticiones las cuales interactúan con elementos que podrían formar parte de otro servicio.

En 4Story existen diferentes servicios tales como el del login (encargado de gestionar el proceso de autentificación de los usuarios), pero también es común crear nuevos servicios para gran cantidad de utilidades, un ejemplo puede ser la implementación de los mencionados anteriormente modos de juego, los cuales suelen funcionar de forma independiente al TMap para poderlos hostear en otros servidores y así escalar un poco la carga que el juego soporta, o bien por cuestiones de seguridad (como por ejemplo, que un error pueda causar un crash absoluto en la parte del servidor).

Para complementar la explicación de los dos servicios del servidor mencionados anteriormente, se proponen a continuación dos ejemplos de uso:

Caso 1: Supongamos que un usuario decide hacer un uso de una acción simple del juego tal y como realizar la venta de un objeto a un NPC (non playable character). Dicha petición sería enviada al servicio TMap, puesto que forma parte del uso corriente del juego. El servidor comprobaría en un primer momento si existe algún tipo de error y enviaría un mensaje al cliente indicando que la venta ha sido fallida, o en caso de ser una venta satisfactoria, podría realizar los cálculos pertinentes en cuanto a la cantidad de dinero obtenido por la venta y enviar una respuesta al cliente.

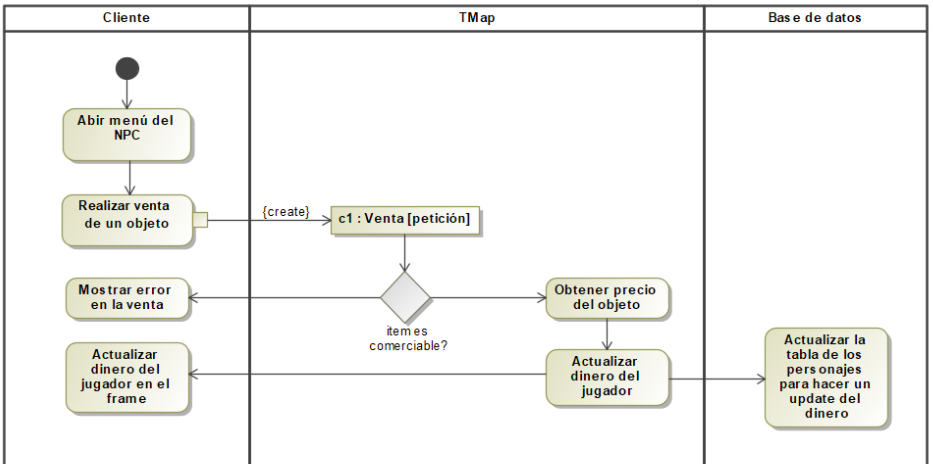

Figura 2. Diagrama de acciones del caso ejemplo 1

Caso 2: Supongamos el caso en el cual un usuario decide invitar a un grupo a otro usuario para así colaborar conjuntamente y destruir enemigos conjuntamente. Esta petición sería recibida por el TMap, el cual dirigiría hacia el TWorld, puesto que los grupos son una característica útil y común en los modos de juego. Este último sería el encargado de gestionarla y devolver al TMap un resultado para informar a todos los jugadores involucrados.

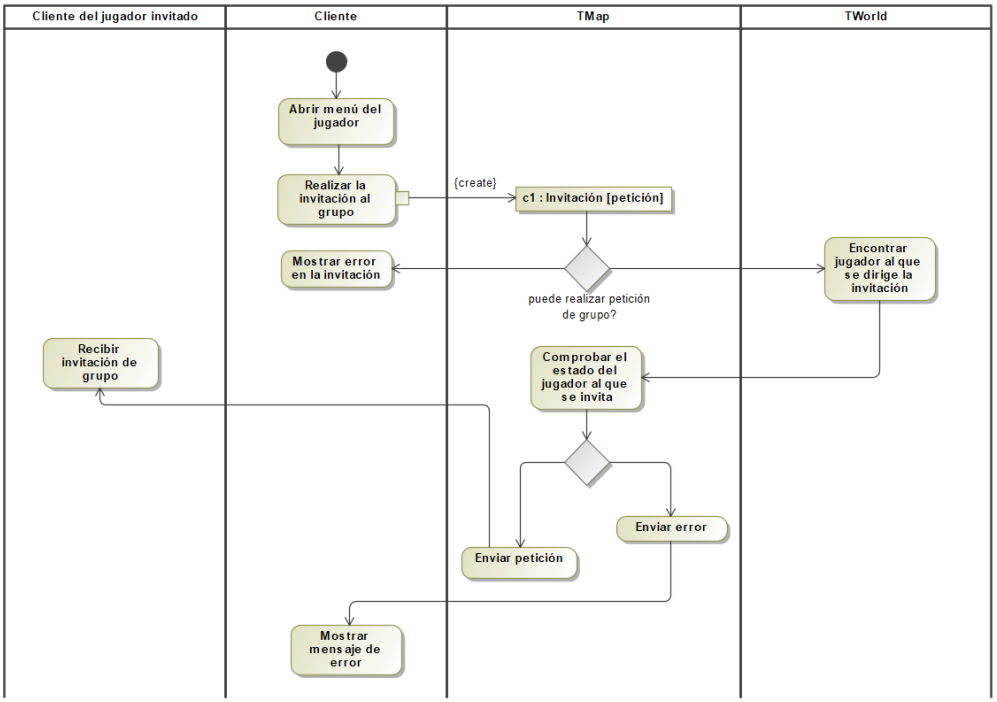

Figura 3. Diagrama de acciones del caso ejemplo 2

Más detalles sobre el funcionamiento en concreto entre cliente y servidor serán detallados en sus correspondientes tareas en caso de que apliquen.

3. Aplicaciones

En el siguiente apartado se detallan algunas de las herramientas que he diseñado para manipular datos de 4Legacy. Hasta el momento han sido un total de 48 las aplicaciones que he creado con dicha finalidad, por lo tanto, se encuentran descritas las más relevantes y las que presentan utilidades innovadoras respecto del resto.

Las aplicaciones tienen como fin realizar la carga de algunos de los datos que afectan al cliente, además, suele desearse crear nuevos datos o borrar/editar los ya existentes. Estos datos acostumbran a almacenarse en código en forma de estructuras las cuales carga el cliente durante la fase de iniciación del juego.

Toda esta información se ubica en archivos binarios de extensión .tcd dentro de la carpeta principal del juego. El cliente acostumbra a realizar una lectura secuencial de los bytes que representan cada uno de los atributos de la estructura para así crear vectores de componentes que el cliente gestionará.

```
void CTChart::ReadTNPC(CArchive& ar, LPTNPC pTNPC)
{
    static TNPC vDummy;

    if (!pTNPC)
        pTNPC = &vDummy;

    DWORD dwMenuID = 0;
    WORD wTempID = 0;

    ar >> pTNPC->m_dwID
        >> wTempID
        >> pTNPC->m_strNAME
        >> pTNPC->m_strTITLE
        >> pTNPC->m_bNPCType
        >> pTNPC->m_bClassID
        >> pTNPC->m_bLevel
        >> pTNPC->m_bCountryID
        >> pTNPC->m_bCollisionType
        >> pTNPC->m_bCanSelected
        >> pTNPC->m_bLand
        >> pTNPC->m_bMode
        >> pTNPC->m_bDrawGhost
        >> pTNPC->m_bDrawMark
        >> pTNPC->m_bDrawName
        >> pTNPC->m_bHouseMesh
        >> pTNPC->m_dwHouseID
        >> dwMenuID
        >> pTNPC->m_dwExecID
        >> pTNPC->m_dwQuestID
        >> pTNPC->m_wItemID
        >> pTNPC->m_dwMaxHP
        >> pTNPC->m_fDIR
        >> pTNPC->m_fPosX
        >> pTNPC->m_fPosY
        >> pTNPC->m_fPosZ
        >> pTNPC->m_fSizeX
        >> pTNPC->m_fSizeY
        >> pTNPC->m_fSizeZ
        >> pTNPC->m_fScaleX
        >> pTNPC->m_fScaleY
        >> pTNPC->m_fScaleZ
        >> pTNPC->m_bCamp;

    pTNPC->m_pTPOPUP = CTChart::FindTPOPUPMENU(dwMenuID);
    pTNPC->m_pTNPC = CTChart::FindTNPCTEMP(wTempID);
}
```

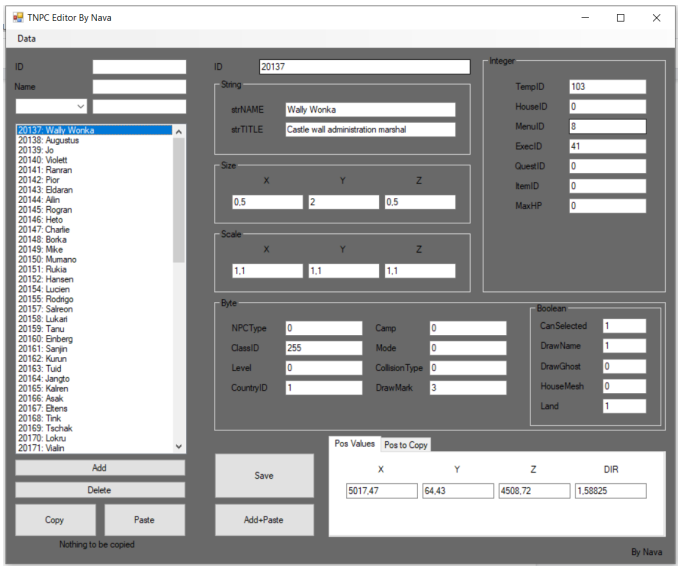

Figura 4. Muestra de cómo el código realiza la carga de los atributos de un NPC

Figura 5. Aspecto visual de la aplicación TNPC

Algunas de las estructuras para las cuales se ha necesitado la creación de aplicaciones han sido: NPCs, habilidades, objetos, niveles, strings, títulos, monturas, razas, etc.

La manipulación errónea del código y de los datos de 4Story ha ocasionado la gestión incorrecta de algunos de los datos que estos archivos contienen, de forma que información que debería de contener la base de datos ha sido definida en muchas ocasiones en este tipo de archivos binarios (los cuales son fácilmente manipulables, ya que cualquier persona con el código público puede observar como este carga las diferentes estructuras y manipular a su gusto cualquier atributo).

Algunos de los componentes mencionados anteriormente, tales como los objetos, contienen información en los .tcd pero también en la base de datos, puesto que son gestionados por el servidor e intervienen en algo más que no aspectos gráficos.

Las aplicaciones que he creado para 4Legacy están diseñadas en .NET Framework, concretamente utilizando Windows Forms Applications en el lenguaje C#. Son de interfaz comúnmente sencilla, puesto que se buscaba realizar una gran cantidad de aplicaciones en un tiempo eficiente.

3.1. TRace

Antes de entrar en detalle en algunas de las aplicaciones más complejas, se describe el funcionamiento de la aplicación TRace en este apartado con el fin de aclarar el aspecto, diseño y funcionamiento general de la mayoría de las aplicaciones.

Surge la necesidad de crear una aplicación si es que se desea modificar o interactuar con algunos de los datos que esta clase contiene. En este caso, se desea tener de cara a un futuro la posibilidad de crear una nueva raza. La aplicación TRace, se encarga de leer el fichero "TRace.tcd" y representar las instancias obtenidas.

Para realizar una interfaz simple que permita visualizar estos datos, es suficiente con la disposición de:

- MenuStrip: Cumple con la finalidad de aportar un menú para realizar la carga de los archivos.
- Listbox: Útil para previsualizar la id de cada elemento y permitir seleccionar cualquiera de estos.
- Textbox: Caja que almacena un atributo perteneciente a la clase.
- Labels: Diferencian con texto los atributos a los cuales hace referencia cada textbox.
- Groupbox: Organizan los atributos según un criterio (en la mayoría de las aplicaciones se usan para diferenciar si el dato es un byte, int, string o float).

Para permitir a esta interfaz opciones tales como borrar, añadir, copiar, pegar o guardar elementos, es suficiente con añadir botones específicos para cada una de estas tareas.

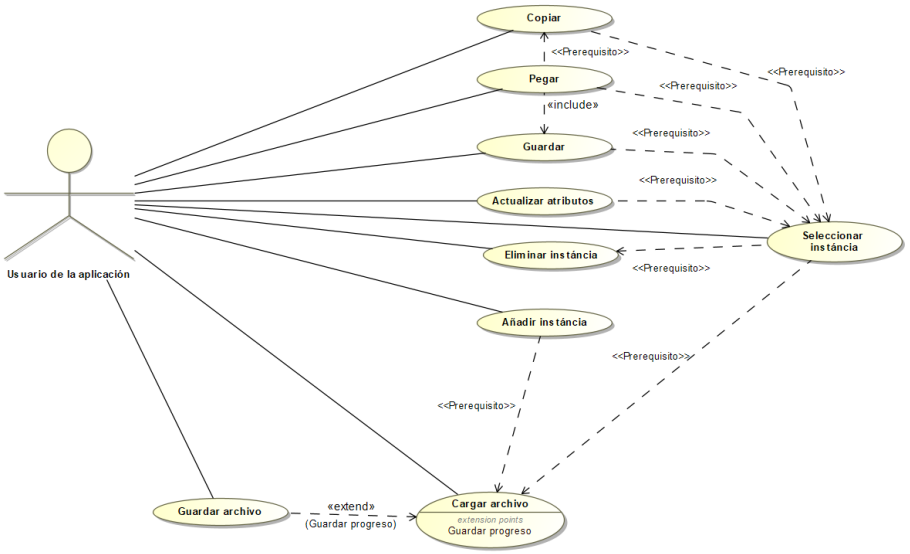

Figura 6. Diagrama de casos de usos de la aplicación TRace

Para la gestión del código es suficiente con dos clases principales, una correspondiente al TRace (con los atributos, los setters y los getters) y otra correspondiente a la interfaz, la cual determinará los eventos que sucederán una vez se interactúe con cada uno de los elementos de esta.

loadMenuStrip_Click: Es la función correspondiente a abrir un diálogo que permite seleccionar un archivo .tcd del cual leer los datos (como estamos tratando la aplicación TRace, comúnmente el archivo a leer tendrá el nombre TRace.tcd).

Si el archivo se ha podido abrir correctamente, lee un natural de 16 bits para asegurarse de cuantas instancias de TRace contiene el archivo. A continuación lee de forma secuencial los diferentes atributos que TRace contiene (en este caso, bRACE, bSEX, dwOBJID y fSCALE). Generalmente, las primeras letras de cada atributo permiten asegurarnos rápidamente del tipo de variable que estos utilizan (siendo b = Byte, f = Float, dw = Unsigned Int, w = Int, str = String). Cada una de estas instancias es guardada en una lista de TRace.

Finalmente, acostumbro a usar una ordenación para mejorar la visualización de los elementos dentro del listbox, en el cual se añade para cada instancia el atributo que permite identificar cada TRace (en este caso dwOBJID es el atributo que los indexa).

saveMenuStrip_Click: De la misma forma que la función comentada anteriormente, debe de encargarse de mostrar un diálogo que permita decidir el path en el cual guardar el archivo generado.

A continuación, basta con realizar el procedimiento inverso a la carga para almacenar los datos en dicho .tcd (se guarda, por lo tanto, primeramente un natural de 16 bits con la cantidad de instancias de TRace y a continuación cada instancia con sus atributos ordenados de la misma forma que a la hora de leerlos).

listBox_SelectedIndexChanged: Permite siempre que haya una instancia de TRace seleccionada en el listbox realizar la carga de sus atributos en los textbox. Basta con realizar listbox.SelectedIndex para saber a qué posición de la lista cargada anteriormente debemos de dirigirnos.

saveButton_Click: Lee los textbox del TRace seleccionado y permite actualizar la lista. Es necesario que no se actualice dicha lista hasta estar seguros de que la conversión de cada dato es correcta (puesto que los TextBox contienen Strings y seguramente queramos convertir estos datos a valores numéricos).

Las funcionalidades de añadir, eliminar, copiar y pegar se llevan a cabo con tareas simples tales como añadir un nuevo elemento a la lista, borrar uno ya existente o crear una instancia temporal con el TRace seleccionado para así pegar los datos en otra instancia.

3.2. TQuest

Tras haber avanzado y creado algunas herramientas simples como la mencionada anteriormente y experimentado con el uso de ellas, surge la necesidad de crear aplicaciones más complejas que permitan manipular datos directamente de la base de datos. Dicha necesidad resulta útil puesto que algunos de los .tcd complementan su información con algunas tablas de la base de datos, y a veces manipular el archivo sin actualizar esta resulta poco práctico.

TQuest es la aplicación que permite enlazar el archivo que hace referencia a las misiones de 4Legacy (llamado “TQuest.mpq”) conjuntamente con todas las tablas de la base de datos que complementan dicha información. La manipulación de esta información

permite crear un sistema de misiones completamente nuevo, el cual permite a los jugadores corrientes de 4Story disfrutar de una experiencia nueva a la hora de alcanzar el nivel máximo y no, en cambio, seguir la corriente historia a la que están acostumbrados.

Para avanzar con el diseño de esta es necesario comprender diferentes factores. La cantidad de misiones que existe en 4Story es elevada, además suele ser una clase con más atributos y relaciones de lo normal, es por eso, que realizar llamadas a la base de datos de forma remota puede resultar poco eficiente a la vez que puede suponer un gran tiempo de espera. En un inicio, con el fin de implementar la aplicación, necesité la conexión remota para así poder disponer de pruebas en un entorno real, para esto, generalmente limitaba la cantidad de misiones que el TQuest cargaba en 100 (puesto que con estas ya tenía una cantidad de datos suficiente para realizar una fase de pruebas temprana y sencilla). A efectos prácticos y futuros, lo que se busca es crear una aplicación con conexión local, la cual permite manipular las misiones internamente en el servidor en el cual se encuentra ubicada la base de datos para así evitar grandes retrasos y por efectos de seguridad.

De cara al diseño de la aplicación, lo primero es estudiar e investigar los datos que interfieren en el TQuest y todos aquellos con los que se desea interactuar:

-Clases de misiones: Esta clase sirve para permitir agrupar las misiones según un nombre. Esto permite a los usuarios de la aplicación encontrar más fácilmente todas las misiones que siguen un mismo hilo, al igual que a los jugadores les permite reconocer a qué secuencia de misiones pertenece la misión que van a realizar.

Por lo general, como en todos los MMORPG, existe un camino a seguir hasta alcanzar el nivel máximo, determinado por los objetivos que las misiones marcan, es decir, suele suceder que una misión te conduzca a un lugar donde luego tendrás que realizar una tarea y moverte hacia otro para seguir con la siguiente. Esta clase es la que permite diferenciar esta secuencia de misiones respecto a otras, para así no confundir al jugador y aceptar misiones que pertenecen a tareas secundarias.

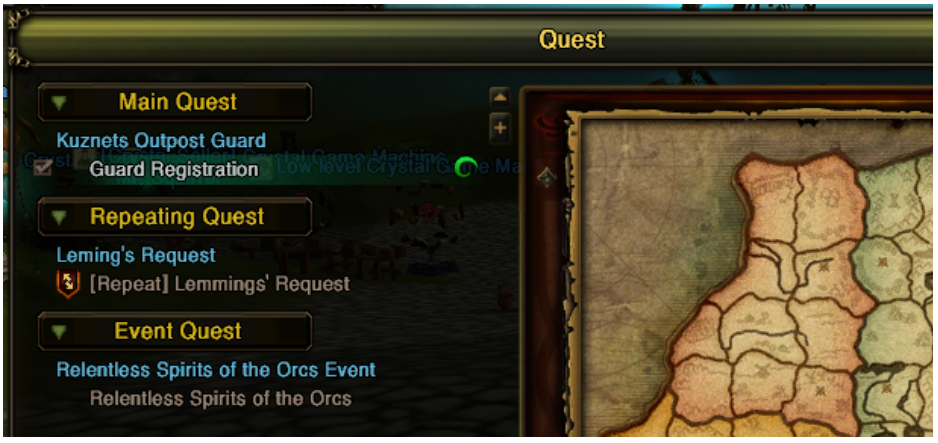

Figura 7. Interfaz de cómo el usuario visualiza una misión y su clase

-Misiones: Contiene toda la información referente a cada misión existente en el juego, como por ejemplo, los niveles mínimos y máximos que se necesitan para aceptarla, los requisitos para completarla o las recompensas.

En esta clase encontramos que aproximadamente la mitad de la información es proporcionada por el archivo binario, mientras que la otra mitad es necesaria de obtener a través de la base de datos.

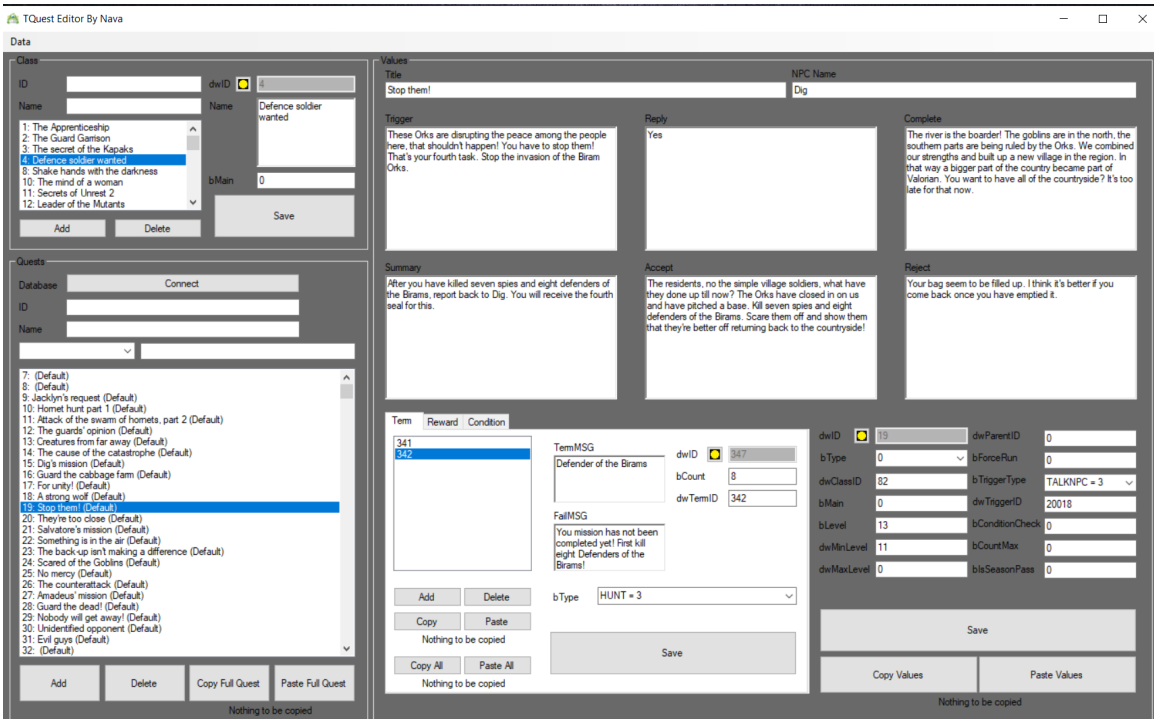

Figura 8. Aspecto visual de la aplicación TQuest

En general, la clase misión dispone de una información maestra y relacionada directamente a ella, la cual define algunas características, tales como una descripción, un diálogo el cual permite crear una historia en el juego, etc. En cambio, también dispone de las relaciones con:

- Term: Definen las diferentes acciones que se deben de realizar para completar la misión. Ejemplos: Destruir a 5x jabalíes, hablar con otro NPC o haber utilizado un objeto previamente.
- Reward: Contiene la información referente a las recompensas que se van a entregar. Ejemplos: Una espada de nivel 50, 5x pociones de vida o dinero.
- Condition: Verifican que ciertas condiciones se hayan cumplido a la hora de completar la misión. Ejemplos: Que el personaje pertenezca a uno de los dos reinos, que disponga de un cierto objeto que ha debido de recoger u obtener previamente o que sea un arquero.

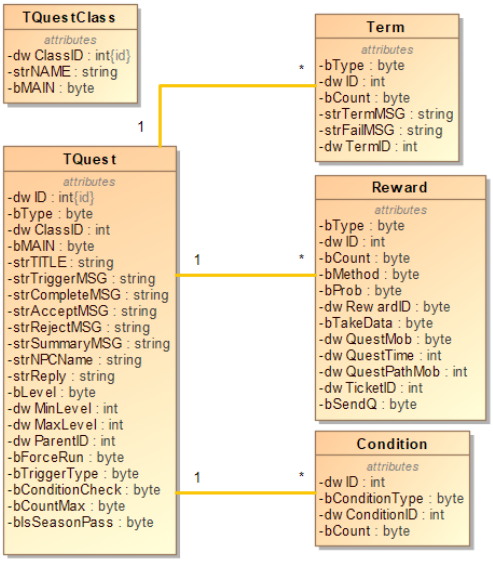

Figura 9. Diagrama de clases respecto a las misiones y sus relacionados

La información exacta y precisa sobre las opciones y la utilidad que proporciona cada atributo es estudiada por Enrique, quien es el usuario principal de dicha aplicación. Para la creación y diseño de la aplicación basta con permitir la carga y la manipulación de los datos de forma correcta.

Se desea, por lo tanto, realizar acciones tales como:

- Conectar la aplicación a la base de datos.
- Permitir la carga y el guardado de los datos en el archivo TQuest.mpq.
- Permitir la carga y actualización de la información en la base de datos.
- Visualizar la información contenida en ambos base de datos y archivo.
- Añadir, eliminar, copiar y pegar nuevas instancias de clases de misiones, misiones, términos, recompensas y condiciones.
- Permitir realizar búsquedas de misiones y clases de misiones según su atributo identificador o el nombre de estas. Adicionalmente, las misiones añadirán otros filtros útiles para encontrarlas según otros criterios.

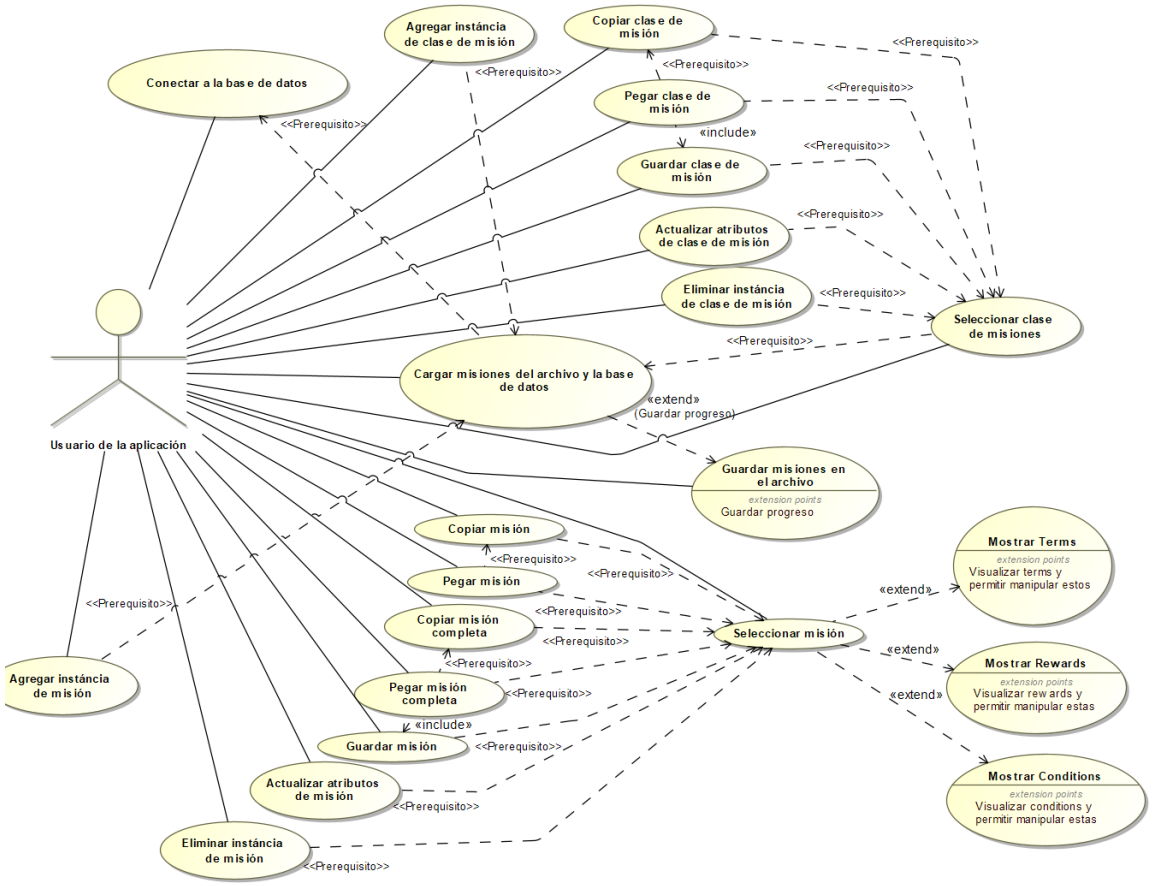

Figura 10. Diagrama de casos de uso de la aplicación TQuest

A continuación se detalla de forma sencilla algunos detalles sobre la implementación de la aplicación:

Para el almacenamiento de las diferentes estructuras, es necesario generar las siguientes clases con todos los atributos, getters y setters.

- Reward
- Term
- TQuestClass
- TQuest
- Condition

Las misiones TQuest son quienes mantienen las relaciones con term/reward/condition, por lo tanto, es importante definir dentro de esta clase una lista de las clases Term, Reward y de Condition.

Por otra parte, así están diseñadas las funciones de carácter básico en la aplicación:

connectButton_Click: Permite al usuario realizar la conexión a la base de datos. La información sobre dicha conexión se encuentra hardcodeada para no permitir flexibilidad, puesto que en muchos casos este tipo de aplicaciones se decide vender, transferir o incluso de algún modo se consigue extraer entre equipos de desarrolladores de 4Story. Para realizar la tarea de conectarse basta con hacer uso de la clase SqlConnection que nos proporciona el espacio de nombres System.Data.SqlClient (proveedor de datos de .NET para SQL Server).

Como se usa la aplicación de forma local, basta con introducir los credenciales de la base de datos de la forma siguiente:

Data Source=<nombre_sqlserver>;Initial Catalog=<nombre_db>;Integrated Security=True

- Data source: Nombre de la instancia de SQL Server a la que se va a conectar.
- Initial Catalog: Nombre de la base de datos.
- Integrated Security: En modo true para que las credenciales de cuenta de Windows actuales se usen para la autenticación.

loadMenuStrip_Click: Al igual que en TRace, inicialmente se necesita del uso de un diálogo para permitir al usuario seleccionar el archivo TQuest.mpq para importar datos. Además, como la aplicación complementa los datos de forma automática con la información de la base de datos, es importante que llegados a este paso se haya realizado la conexión con esta.

Una vez seleccionado el archivo, la aplicación realiza la carga de las clases de misiones (de la misma forma que se realiza la carga de las TRace), las cuales se encuentran completamente descritas en el TQuest.mpq.

A continuación, es necesario cargar las misiones, a las cuales hay que añadirles llamadas a la base datos para seleccionar los atributos que no proporciona el archivo.

Cada misión contiene en el archivo de forma secuencial sus terms, rewards y conditions, de forma que una vez leído el último atributo de misión, introduce en un natural de 32 bits la cantidad de terms que hay, y a continuación todos los atributos de term. Del mismo modo, una vez finalizados los terms, encontramos en un natural de 32 bits la cantidad de rewards que contiene el archivo, seguidamente de los atributos de reward. Lo mismo pasa con las conditions.

Es necesario, por lo tanto, que cada vez que se lea una instancia de alguna de estas relaciones, se añada a la lista que contiene la clase TQuest.

saveMenuStrip_Click: A la hora de almacenar los datos, también se distinguen dos fases. En la primera, se guarda en el archivo TQuest.mpq las clases de misiones, comenzando por un entero de 32 bits indicando cuantas instancias existen y añadiendo de forma ordenada los atributos de cada instancia.

Lo siguiente es realizar el almacenamiento de las misiones, las cuales, además de encontrarse en el archivo mencionado anteriormente, también disponen de información en la base de datos.

Esta tarea se simplifica bastante, pues solo es necesario que la función actualice el fichero de la misma forma que hemos leído la información pero, esta vez a la inversa (es decir, creando el nuevo archivo TQuest.mpq resultante de escribir las listas de datos).

El hecho de realizar un guardado del fichero mediante el menú superior de la aplicación proporciona que el usuario no deba de estar esperando cada vez que actualiza un dato a que se genere el archivo de salida, sino que lo hará una única vez cuando desee guardar el progreso. Por otro lado, la base de datos sí permite la actualización dinámica de esta, de forma que cada vez que se guarda el progreso para una instancia, ya podemos recurrir a un update de la fila de la base de datos, proporcionando así, que a la hora de guardar la aplicación ya se encuentre correctamente actualizada.

listBoxQuest_SelectedIndexChanged: Cada vez que se selecciona una misión diferente dentro del listbox es necesario que también se actualicen las listbox de sus relaciones, por lo tanto, no basta con añadir la información a cada textbox, sino que

además hay que añadir a listboxTerm, listboxReward y listboxCondition las instancias que contiene la misión seleccionada de cada uno de estos elementos.

saveQuestButton_Click: Para realizar el guardado de una instancia de TQuest, es necesario recoger la información de los textbox y convertirla para así actualizar el listado. Además, se debe de realizar un update en la tabla de la base de datos para guardar los datos.

Cada clase dispone de su propio botón de save, es decir, que guardar una misión no implica actualizar sus terms, si se desea dicha acción, es necesario hacer clic en el botón de save que contienen los terms.

addQuestButton_Click: El hecho de añadir una misión a la base de datos dispone de una característica adicional resultante de la manipulación incorrecta de la base de datos por parte de desarrolladores de 4Story previos. Actualmente las tablas disponen de índices dispersados, de la cual forma, algunas incluso han llegado al número máximo de su capacidad, es por eso que a la hora de añadir datos siempre realizo algoritmos que busquen préviamente una id libre antes de dar por hecho que la última disponible va a ser también la más grande.

La siguiente tarea a realizar es insertar una nueva fila a la base de datos con la id resultante del algoritmo anterior.

deleteQuestButton_Click: Es suficiente con eliminar de la tabla TQUESTCHART la misión con la dwQuestID seleccionada, de la misma forma que debemos de eliminar todas las filas de TQUESTTERM, TQUESTREWARD, TQUESTCONDITION todas las filas relacionadas con dicha misión.

pasteQuestButton_Click: Pegar una misión con toda su información en otra ya existente conlleva inicialmente actualizar la tabla de misiones de la base de datos, así como el listado de TQuest.

A continuación, se debe de eliminar todas las filas en sus relaciones que apuntaban a dicha misión, puesto que ahora los term, reward y conditions van a verse también afectados. El siguiente paso es actualizar el listado de TQuest con la información de la misión copiada y realizar inserciones a las tablas de sus relaciones buscando ids no ocupadas.

A la hora de actualizar el listado no basta con hacer que el índice seleccionado sea ahora la TQuest temporal que hemos creado a la hora de copiar, sino, que hay que disponer de una función que permita clonar y replicar dicha instancia. En caso contrario, lo que estaríamos haciendo es que el índice seleccionado apunte al índice copiado, generando así que cuando modifiquemos una misión también modifiquemos todas las que apuntan a ella.

El resto de la aplicación proporciona funciones de diseño similar y simplificado pero aplicadas a las clases Term, Reward y Condition.

3.3. TMon

Hasta el momento en el que decidí realizar esta aplicación, la única manera de crear una criatura dentro del juego era mediante la inserción manual en la base de datos. Esto conllevaba un proceso lento y poco eficaz, puesto que en muchos casos las coordenadas no eran del todo precisas y resultaba lioso modificar diferentes tablas a la vez manteniendo la coherencia de los datos.

La aplicación TMon, por lo tanto, agiliza este proceso. Además, hasta el instante no existía una forma gráfica para visualizar las criaturas y sus posiciones desde fuera del

juego, pues el proceso a seguir era reiniciar el servidor y acceder mediante un personaje a la zona la cual se quiere consultar (ya que echando un vistazo a la base de datos es difícil llegarse a imaginar cómo están distribuidos los enemigos). TMon permite de una forma dinámica visualizar los monstruos ya existentes en cualquiera de los mapas de 4Story a la vez que realizar operaciones con estos (cambiar su aspecto, posición, recompensas, etc.).

En 4Story existe un mapa global, el cual representa el conjunto de todas las regiones a las que un jugador puede acceder (la mitad de este pertenece al reino de Valorian y la otra mitad al reino de Derion). Cada una de estas regiones tiene su propio mapa, es por eso que existen gran cantidad de zonas, así como monstruos a gestionar.

Figura 11. Mapa global de 4Legacy con las regiones originales

Figura 12. Mapa de la región Yesode

Las imágenes correspondientes a cada mapa están a disposición de nuestro equipo, de la misma forma que disponemos de las imágenes de los mapas creados y existentes únicamente en 4Legacy (creados con una herramienta externa). Estas imágenes resultan útiles de la misma forma que otras aplicaciones que he creado anteriormente para añadir funcionalidad a la aplicación TMon.

Figura 13. Aspecto visual de la aplicación TMon

TMon permite cargar los mapas de una carpeta local creada por la propia aplicación para así cargar de la base de datos todos los monstruos que se encuentren en dicho mapa. La aplicación permite seleccionar 3 tipos de punteros (select, add y delete), en función del tipo de ratón elegido, el mapa interactúa de una forma distinta al hacer clic.

- Select: A la hora de hacer clic en uno de los puntos rojos que se muestran en el mapa, carga toda su información, resultando útil para realizar consultas y/o copiar su información. En el caso de querer modificar alguno de los atributos de un monstruo existente, es suficiente con seleccionar este, actualizar el textbox y hacer clic en guardar.
- Add: Cada vez que se utiliza este puntero sobre el mapa, se crea un monstruo con la información seleccionada actualmente (por lo tanto, es importante haber seleccionado anteriormente un monstruo para cargar sus datos, o bien haber rellenado los textbox manualmente). Además, el mapa se vuelve a colorear indicando un punto rojo en el lugar donde se ha invocado la nueva criatura. La aplicación gestionará automáticamente la base de datos para hacer insertar y actualizar los nuevos datos.
- Delete: Elimina de la base de datos toda la información referente a cada criatura la cual se haga clic con este puntero. Además, la aplicación retornará el color original del mapa para hacer desaparecer el punto rojo del monstruo eliminado.

Esta herramienta ha supuesto que añadir monstruos a un mapa completo suponga una tarea de 30 minutos, mientras que anteriormente podía ser un proceso de más de 12 horas.

En cuestión de diseño, esta aplicación cuenta con las siguientes tablas:

- TMONSPAWNCHART: Cuenta con toda la información sobre los monstruos que deben de aparecer en cada punto del juego. Es importante, por lo tanto, que contenga información sobre el mapa, la coordenada y la dirección sobre la cual se va a realizar la aparición, al igual que otros datos complementarios.
- TMAPMONCHART: Esta tabla relaciona cada aparición (spawn) con el tipo de monstruo que es. En la aplicación cumple con la función de permitir consultar a cada punto que tipo de monstruo se le ha asignado. Además, este puede ser modificado de forma que escribiendo el nombre se generan las posibles opciones disponibles para autocompletar el monstruo.
- TMONATTRCHART: Cada monstruo dispone de un Attr, el cual sirve para englobar sus características, los monstruos de un mismo nivel suelen por lo general tener un mismo Attr, el cual define rasgos como la vida, el daño o la velocidad de este.
- TMONITEMCHART: De forma adicional, cada monstruo cuenta con una cantidad de posibles recompensas en forma de objetos, esto hace que los jugadores se vean motivados a cazar y destruir todo tipo de criaturas.

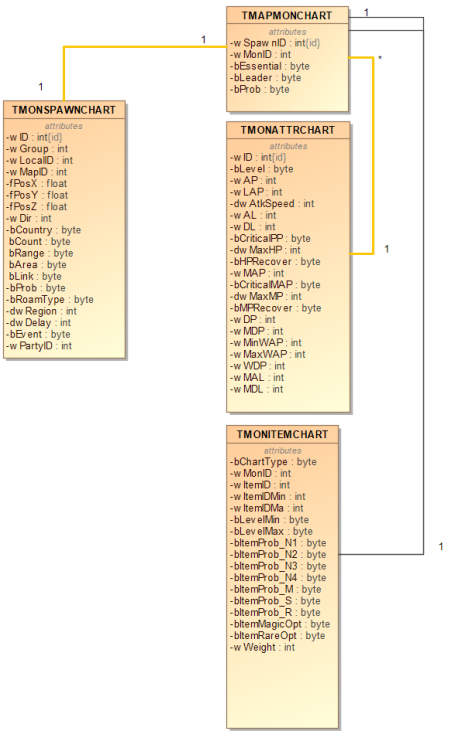

Figura 14. Diagrama de clases respecto de las apariciones de monstruos y sus relacionados

Esta aplicación, por lo tanto, no necesita del uso de ningún archivo y únicamente permite la gestión de información de la base de datos. Esto proporciona que el código no necesite de la disposición de clases para almacenar información, ya que esta es temporal y se encuentra almacenada en los diferentes componentes de la aplicación (es decir, podremos rellenar la información a cerca de un monstruo a la hora de guardarlo simplemente convirtiendo el texto de los textbox al valor que espera recibir la base de datos).

A continuación se detalla una breve descripción de cómo están implementadas algunas de las funciones más relevantes dentro de la aplicación:

loadMenuStrip_Click: De forma automática, accede a la carpeta donde se encuentra la aplicación en busca de los siguientes archivos:

- Path.txt: Si no se encuentra este path, abrirá un diálogo al usuario para que este seleccione la ubicación del archivo TMinimap.tcd, una vez seleccionado, se creará el archivo Path.txt con la dirección de este .tcd para no preguntarlo de nuevo en un futuro. Este .tcd contiene información sobre los mapas que resulta útil en el presente de la aplicación, pero también de cara a un futuro.

  Una vez abierto el archivo mencionado, la aplicación realizará la lectura de todos los datos dentro de este y los guardará en un listado de la clase TMinimap.

- IDs.txt / Customs.txt / Names.txt: Estos tres archivos se generan automáticamente y sirven como base de datos sencilla y local para almacenar la información sobre los mapas que se registran en la aplicación.

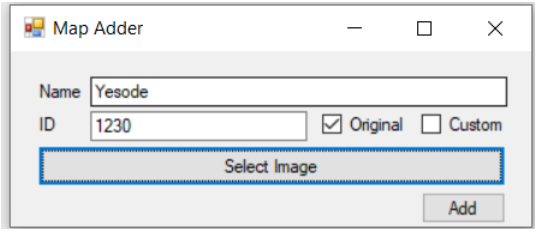

Figura 15. Aspecto gráfico del menú para añadir mapa en la aplicación TMon

Mediante el menú mostrado anteriormente, el usuario puede dar de alta mapas en la aplicación, los cuales actualizarán los ficheros anteriormente comentados y se cargarán a partir de ese instante.

comboBox_SelectedIndexChanged: Esta función se encarga de realizar la carga del mapa seleccionado basándose en la opción seleccionada.

Primeramente se realiza una consulta en la carpeta Img para encontrar la imagen que pertenece al mapa seleccionado (esta imagen habrá sido copiada en esta carpeta de forma automática cada en el instante de creación del mapa dentro de la aplicación). En caso de encontrar dicha imagen, se muestra.

La siguiente tarea que ejecuta la aplicación es consultar en la base de datos todas las criaturas que pertenecen al mapa seleccionado. Para realizar dicha tarea, cabe considerar que los mapas originales obtienen todos id = 0, mientras que los personalizados dotan de un identificador único. Por lo tanto, para encontrar criaturas de un mapa original, debemos de asegurarnos que id = 0 y que, además, las coordenadas cumplan con las condiciones siguientes:

$X + 1024 < coord\_x < X \;\; and \;\; Z + 1024 < coord\_z < Z$

Dónde X, Z representan las coordenadas del mapa, obtenidas gracias al archivo TMinimap.tcd cargado anteriormente, el cual entre otras cosas, determina la posición final de un mapa, el cual, es de tamaño 1024 x 1024.

Por lo tanto, concluimos que los mapas personalizados tienen X = 1024 y Z = 1024, y que los mapas originales van uno a continuación del otro (es decir, están entrelazados, hecho que causa que podamos andar de uno al otro de forma dinámica).

A continuación se muestra un ejemplo ficticio en el cual se describe respecto 2 regiones del mapa original de 4Story, cual sería el rango de coordenadas que debemos de buscar para encontrar los monstruos de Horusland o Tarat.

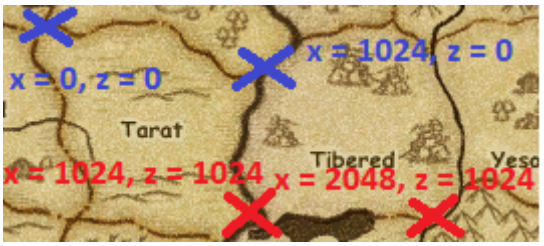

Figura 16. Ejemplo sobre las coordenadas x, z

Por lo tanto, los monstruos de Tibered cumplen con id = 0, 1024 < x < (1024 + 1024), 1024 < z < (1024 + 1024).

Finalmente, tras haber recogido todas las criaturas de este mapa gracias a la llamada a la base de datos, la función se encarga de crear un bitmap con la imagen, donde en cada coordenada X, Z de aparición de monstruo es dibujado un punto de dimensión 5x5 píxeles para que resulte más visible y sencillo de hacer clic.

Además, esta función crea un vector de coordenadas con el conjunto de puntos de aparición de monstruos que existe en el mapa cargado. Este listado nos permitirá de forma eficiente consultar cada vez que se interactúe con el mapa si existe alguna criatura en dicha posición.

pictureBox_Click: Para gestionar las interacciones del usuario con el mapa se necesita consultar en un inicio el modo en el que se encuentra el puntero.

En el caso en el cual deseamos seleccionar o borrar criaturas, es necesario consultar previamente si la coordenada indicada coincide con la correspondiente a alguna criatura. Para realizar dicha tarea, la aplicación accede al vector de coordenadas creado en la función anteriormente descrita. Con tal de permitir al usuario una precisión más amplia, no es suficiente con verificar si coincide exactamente la coordenada del ratón con la del monstruo, pues también miraremos si 2 píxeles en cada dirección existe algún monstruo (esto acabaría formando una hitbox de 5x5 píxeles, es decir, lo equivalente a uno de los puntos rojos). De esta forma nos aseguramos que clicando cualquier extremo de un punto rojo, la función detectará su objetivo.

Para crear nuevos puntos de aparición no es necesario comprobar nada, pues se permite crear diferentes monstruos en una misma coordenada.

Según el modo en el que se encuentre el puntero deberemos de realizar llamadas a la base de datos en forma de consulta, de borrado o de inserción. Además, es importante añadir un nuevo punto o eliminarlo del mapa en el caso de crear o eliminar criaturas.

monName_TextChanged: Se llama a esta función con la finalidad de autocompletar el nombre de un monstruo el cual se desea actualizar. En el caso de que se cumplan ciertas condiciones, tales como que haya un mínimo de 3 caracteres introducidos en el textbox, este método se encarga de obtener de la tabla TMONSTERCHART todos los monstruos cuyo nombre contiene la cadena introducida.

Sí, por lo contrario, el texto que se ha introducido coincide con algún monstruo, se corregirá el textbox MonID. Esto permitirá que mediante un texto autocompletado no sea necesario acceder a la base de datos para consultar la id del monstruo al que corresponde.

3.4. Launcher

A medida que el tiempo ha avanzado, el juego se ha encontrado más a punto para su lanzamiento oficial, de forma que era necesaria la creación de un lanzador, el cual, aparte de otras necesidades, cumple con la funcionalidad de permitir a los usuarios actualizar el juego a su última versión.

La modificación de archivos ubicados en la carpeta principal del juego 4Legacy tales como los .tcd conlleva la necesidad de crear un sistema de versiones, el cual permita a los usuarios actualizar su carpeta al estado de despliegue oficial. Los archivos son modificados constantemente, puesto que esto permite crear nuevo contenido o modificar el ya existente. Es común que al menos una vez al mes exista la necesidad de actualizar tcds para realizar balance en el juego y otras tareas constantes.

Como resultado a esta necesidad, surge la creación del llamado Launcher, el cual cumple con las siguientes funcionalidades:

- Permitir actualizar los archivos pertenecientes a una misma versión.
- Permitir avanzar a la última versión.
- Permitir consultar el estado de la descarga
- Dar la posibilidad al usuario de elegir entre la interfaz de juego antigua (la cual se utilizó en 4Story durante los 8 primeros años) o la nueva interfaz.
- Informar sobre las redes sociales así como la página web de 4Legacy.
- Mostrar las últimas noticias que contiene la web, de forma que se indique información básica sobre cada una de ellas y se redireccione el usuario a la web en caso de desear más detalles.
- Indicar el estado actual del servidor (online, offline o en mantenimiento).
- Habilitar al usuario para que pueda entrar al juego si es que este ha conseguido actualizar de forma satisfactoria su carpeta y el servidor se encuentra disponible.
- Ofrecer la opción de realizar una reparación de los archivos, resultando así en la actualización completa de todas las versiones hasta el punto oficial.

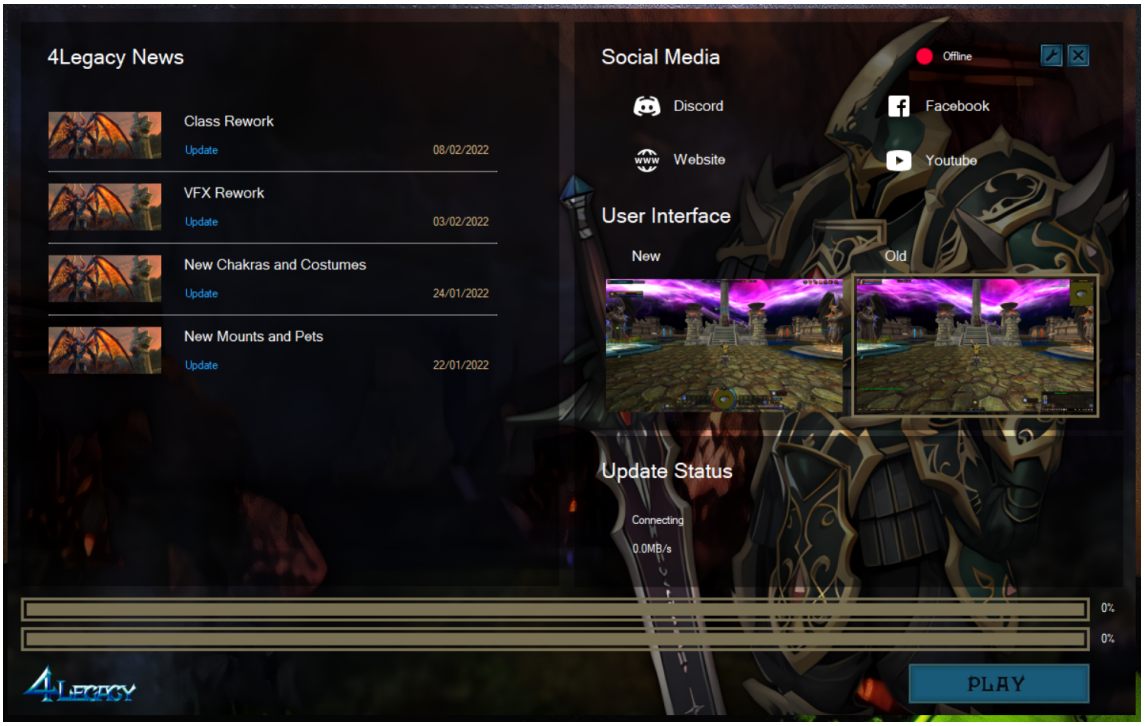

Figura 17. Aspecto visual de la aplicación Launcher

La aplicación cuenta con dos barras de carga. La superior indicará el progreso de la actualización a lo largo de las versiones, es decir, que si existen 2 y ya disponemos de todos los archivos de la primera, se mostrará completa al 50%. La barra inferior muestra el progreso de la actualización del actual parche, siendo determinada por el número de archivos que queda por descargar, es decir, que si la versión que nos encontramos cuenta con 10 archivos y hemos progresado con 3 de ellos, se mostrará el 30%.

Para la creación de esta aplicación he realizado uso de la misma tecnología que hasta ahora, puesto que el dominio de la herramienta supone una facilidad y capacidad de creación extra. En este caso, el diseño y la parte estética sí debe de ser acogedora, pues este lanzador es la carta de presentación a los usuarios.

Cabe destacar que el Launcher debe de realizar una conexión a un servidor, en el cual se encontrará hosteado y desplegará el lanzamiento de los archivos demandados. Con el fin de permitir flexibilidad de cara a una futura actualización del Launcher, he creado el llamado Updater, el cual será introducido más adelante en la documentación.

Seguidamente, se muestra el proceso a seguir por parte de la aplicación a nivel de diseño e implementación.

Durante el despliegue de la aplicación, esta se encarga de obtener la dirección IP a la cual debe de realizar todas las peticiones que comentaremos a continuación. Para obtener esta IP (la cual es importante considerar que es posible que cambie en un futuro, por ejemplo, por la contratación de un servidor distinto) se realiza una llamada a la API de 4Legacy.

La respuesta JSON tiene como función devolver tanto el link de Mega, Drive y la IP del launcher.

```
{
    "mega": "",
    "googledrive": "",
    "launcher": ""
}
```

El Launcher se encarga de deserializar este objeto y crear los diferentes strings que permitirán realizar peticiones al servidor:

```
"http://" + downloadList.launcher + "/launcher/", RegistryManager.GetUI());
"http://" + downloadList.launcher + "/launcher/patch.php?version=";
```

El primer string permite realizar descargas de archivos del servidor mediante la creación de una URI para acceder a la carpeta /launcher/old o /launcher/new, en función de la interfaz que se haya decidido usar. Dentro de ambos archivos encontramos el ejecutable correspondiente al juego llamado TClient.exe. Ambos se sitúan en su carpeta correspondiente cada vez que se compila el código del cliente de 4Legacy.

Por otra parte, también se dispone de las siguientes URI /launcher/<old/new>/<version>/<archivo> para poder acceder a la descarga de un archivo perteneciente a cualquiera de los parches. Se diferencian las versiones en función de la interfaz usada, puesto que a veces es posible que algún .tcd se vea modificado de forma distinta para una de ellas.

El segundo string permite crear una URI a la cual se le añade la versión la cual dispone el usuario del juego (la última versión que un usuario ha descargado es guardada a

la hora de actualizar en un registro de su ordenador). Dicha URI permite obtener un JSON, con información esencial sobre las versiones que quedan por descargar (indicando para cada versión los archivos y el path donde deben ubicarse estos en la carpeta principal de 4Legacy).

Una vez el Launcher se ha inicializado, llama a esta URI para indicar si el usuario se encuentra en la última versión, o necesita descargar ficheros.

En el caso de encontrarse en la última versión, es suficiente con actualizar los componentes visuales. Por lo contrario, si hay que actualizar los archivos, se sigue el proceso descrito a continuación:

Para cada versión obtenida en el último JSON deserializado comentado anteriormente se actualizan los componentes visuales (controlando la barra de progreso y el texto indicando en qué situación se encuentra).

Además, se debe acceder a la API para que realice de forma asíncrona la descarga de cada uno de los archivos que componen la versión.

Todos estos archivos se sitúan de forma temporal en una carpeta llamada temp dentro de la carpeta 4Legacy.

Durante el progreso de descarga de estos se realiza un cálculo sencillo sobre la velocidad de descarga, en función de la cantidad de bytes recibidos y el tiempo transcurrido.

Una vez transcurrida la descarga de cada uno de estos archivos se traslada de la carpeta temporal a su path final. Además, se comprueba el estado de la versión, comprobando si se debe de avanzar a la siguiente, o, por lo contrario, se debe de seguir por el siguiente archivo.

Se detalla a continuación el funcionamiento de algunos métodos de la aplicación.

playButton_Click: Para acceder al juego se verifican dos condiciones, la primera consiste en consultar si el servidor se encuentra disponible. Para realizar esta acción se llama a la API de 4Legacy, obteniendo un entero el cual resulta en 1 si el servidor se encuentra operativo para los usuarios.

La segunda condición que se verifica (a un nivel muy simple) es consultar si alguno de los procesos que el usuario tiene abierto coincide con el ejecutable del juego.

Finalmente, en caso de superar ambas condiciones, se abre el ejecutable del juego enviando por parámetro la IP a la que se deben de conectar y el puerto. Estos campos hacen referencia al servidor que hostea el juego.

getNews: Esta función es llamada por la aplicación en el instante de ser inicializada, pues es la encargada de recibir de la web las últimas noticias y actualizar los componentes visuales.

Para obtener las noticias es necesario deserializar un listado de New recibido por la API de 4Legacy. Para cada noticia se indica una id, título, descripción, fecha, estado y autor.

Las noticias están categorizadas según su estado, el cual distingue las noticias según si son de mantenimiento, eventos, mantenimiento, actualizaciones, anuncios y otros. De este modo, las noticias con un mismo estado reciben la misma imagen (esta cuestión de diseño se ha tomado por el simple hecho de no disponer de diseñador gráfico para crear un

banner personalizado para cada noticia). Para obtener la imagen que representa cada estado, se accede al path https://4legacy.online/assets/images/banners/<estado>.png.

3.4.1. Updater

El Launcher cumple con la funcionalidad de actualizar el juego, pero ahora bien… ¿Qué sucede si se necesita actualizar el Launcher?

El updater es una aplicación la cual pretende ser lo más simple posible para cumplir con un diseño básico y flexible. Su funcionalidad es realizar la actualización del Launcher.

El propio Launcher es el encargado de lanzar la ejecución del Updater. Esta acción sucede a la hora de iniciar el Launcher, de forma que se comprueba si la fecha de modificación del lanzador que hay ubicado en la carpeta /launcher del servidor es más actual que la fecha de modificación del lanzador que dispone el usuario.

En el caso de necesitar actualizar el Launcher, este abrirá un proceso con la aplicación Updater.exe que se encuentra ubicada por defecto en la carpeta principal de 4Legacy (en caso de no encontrarse, el propio Launcher gestionará las llamadas necesarias al servidor para descargarlo). A continuación, el Launcher cerrará el proceso, de forma que el actualizador pueda descargar la última versión sin solapar con este.

Figura 18. Aspecto visual de la aplicación Updater

4. Cliente y servidor

A lo largo de este fragmento de la documentación, se describe el proceso a seguir para implementar diferentes tipos de características dentro del juego.

4.1. Daily frame

4Story carga los componentes gráficos pertenecientes a menús, marcos, botones y algunos textos basándose en uno de los ficheros binarios, llamado TClientcmd.tif. Disponemos de una herramienta para poder manipular este fichero, la cual es fundamental en el caso de querer acceder a la edición de cualquier frame (el concepto frame corresponde en 4Story a cualquier menú gráfico, el cual se genera como resultado de alguna acción, como por ejemplo interactuar con un NPC o clicar un botón).

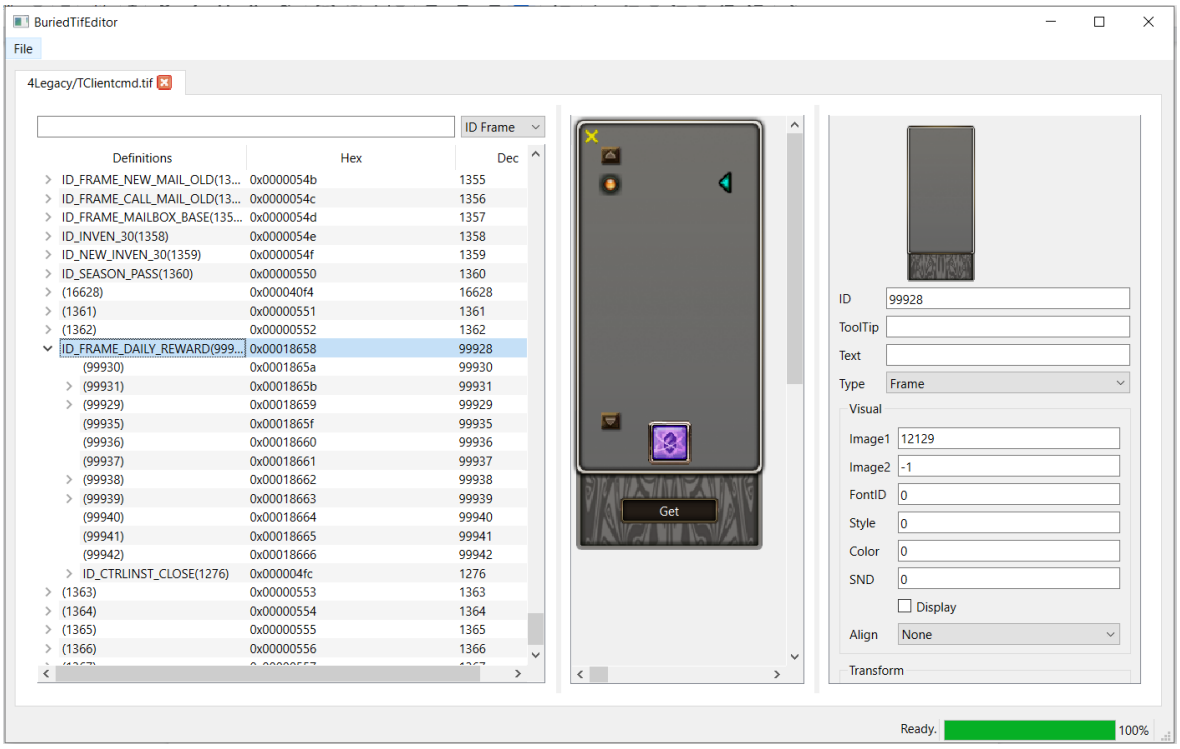

Figura 19. Aspecto visual del Daily frame en la aplicación que lee el TClientcmd.tif

El uso de esta aplicación para crear el frame con todos sus componentes es lo primero que se debe de realizar (es posible que en una primera versión sea difícil de imaginar y posicionar todos los elementos que se usarán, pero podemos modificar el archivo de forma progresiva).

En esta documentación no se describe el uso y la forma de añadir cada tipo de componente al frame, pues solo se ha necesitado hacer de su uso para crear el aspecto visual del frame.

El frame que se desea implementar en este caso, corresponde a un menú que se abre al acceder a un personaje, el cual pretende recompensar al jugador por haberse conectado a durante varios días consecutivos.

Algunos de los aspectos a tener en cuenta son los siguientes:

- El frame se abrirá de forma automática cada vez que el jugador acceda a uno de sus personajes (independientemente de si está dentro de la misma sesión o ha realizado logout antes de cambiar a otro).
- Habrá un total de 10 recompensas, las cuales se reiniciarán una vez se alcance la última de ellas.
- Para entregar una recompensa al jugador debe de haber cumplido un mínimo de 30 minutos de juego sin haber realizado desconexión del personaje.
- Cada vez que se entrega una recompensa se registra la fecha para así situar la siguiente recompensa disponible al cabo de 24 horas.
- Si han transcurrido más de 48 horas desde la última conexión, se reiniciarán las recompensas.
- Un usuario puede reclamar las recompensas de los días anteriores si ha cumplido con los requisitos. Para reclamar las recompensas de los días anteriores se debe de hacer de forma secuencial, de forma que para reclamar la recompensa del día 3 es necesario haber reclamado anteriormente la del día 2 y la del día 1.
- Cada recompensa cuenta con la entrega de un solo tipo de objeto, el cual puede entregarse en las cantidades disponibles de un stack (es decir, podemos entregar 10x pociones de vida si es que estas permiten agruparse en cantidades de hasta 10 pociones).
- Se muestran con una redonda de color verde las recompensas listas para recoger, en color naranjas las ya obtenidas y en color gris las que aún no se han obtenido.
- Debe de haber una flecha la cual permita seleccionar los diferentes días para así consultar sus recompensas o reclamarlas en caso de encontrarse disponible.
- El sistema debe de localizar sus variables a la cuenta y no a cada personaje, pues el progreso y las recompensas son comunes en todos los personajes y no pueden recibirse más de una vez.
- Si un usuario consigue cumplir las condiciones suficientes para reiniciar las recompensas al primer día, perderá todos los objetos que no haya reclamado con anterioridad (es decir, es necesario que el usuario reclame las recompensas de los 10 días o las perderá).
- Las recompensas son mutables y, por lo tanto, se encuentran en la base de datos para así poder cambiarlas a conveniencia de los administradores.

Figura 20. Aspecto visual del Daily frame en el juego

Para implementar un frame es necesario la creación de una nueva clase, la cual recibe el nombre de CTItemDailyDlg en este caso. Esta debe de estar correctamente inicializada en la función que inicializa todos los recursos del cliente.

Generalmente, las clases de los frames cuentan con los siguientes elementos:

- De forma privada, cuentan con todos los atributos correspondientes a cada componente del frame, en este caso, cada botón será un atributo, al igual que cada imagen o texto que se desea mostrar.
- Adicionalmente, se añaden los atributos que van a ser útiles para la gestión de esta clase, en este caso en concreto, se necesita guardar el progreso de recompensas diarias del jugador, así que existe una variable que determina el estado en el que se encuentran las recompensas (es decir, indica cuántos días lleva el jugador cumpliendo con los requisitos) y una variable que indica hasta qué día ha reclamado el jugador sus recompensas (es decir, indica qué recompensas ha recogido y qué recompensas quedan pendientes).
- Método constructor.
- Método destructor.
- Una función llamada Render, encargada de gestionar todos los elementos visuales que pueden cambiar de forma dinámica de un tick a otro. En este caso, la función de Render se encargará de mostrar el objeto que se entrega en cada recompensa (puesto que los días también pueden cambiar, pero para que esto suceda se debe de clicar un botón, de la cual forma, su accionador será el botón y no sucederá de forma automática).
- Otros métodos complementarios, los cuales cumplan con necesidades de la clase en específicos, tales como getters y setters.

Por lo tanto, para realizar el funcionamiento de este frame es necesaria la interacción con el servidor, el cual debe de recoger de la base de datos la situación en la que se encuentra el jugador (esta petición se produce cuando un personaje accede al juego). Una vez dicha tarea se haya llevado a cabo, el cliente renderiza el frame con todos sus componentes disponibles.

Tanto cliente como servidor deben de estar preparados para la interacción que el usuario pueda realizar con el frame, el cual ofrece la posibilidad de obtener una recompensa y consultar la recompensa anterior o posterior.

El servidor es el encargado de consultar si alguno de los jugadores activos ha cumplido con los requisitos para avanzar un día en las recompensas diarias y dar un aviso a dicho jugador. Como es una tarea que puede llevarse a cabo en cualquier instante de la ejecución del servidor, se encuentra situada en el timer del jugador perteneciente al TMap.

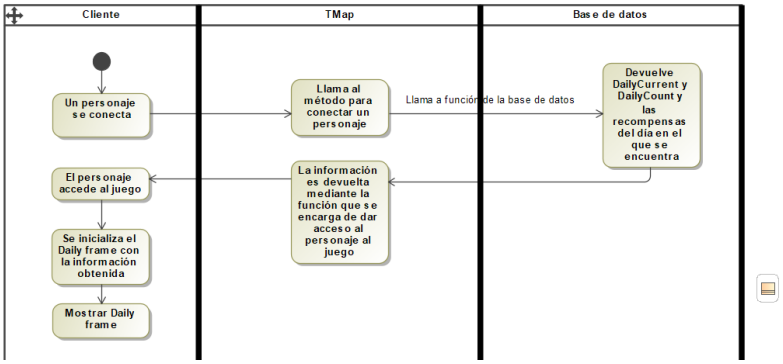

Figura 21. Diagrama de acciones de cómo se inicializa el Daily frame

En el caso anterior, una petición surge con origen servidor y destino cliente, pero esto puede suceder a la inversa, un ejemplo es el caso en el que un jugador reclama alguna de sus recompensas realizando clic en el botón “Get” del frame.

Cada botón tiene asignada una función en el cliente. Esta función representa un comando con una id única, el cual es llamado a la hora de clicar el botón.

En este caso en concreto, el comando es llamado GM_GET_DAILY. Su handler, llamado OnGM_GET_DAILY, es el encargado de comprobar si el jugador contiene algún espacio libre en el inventario para así poder realizar una petición al servidor.

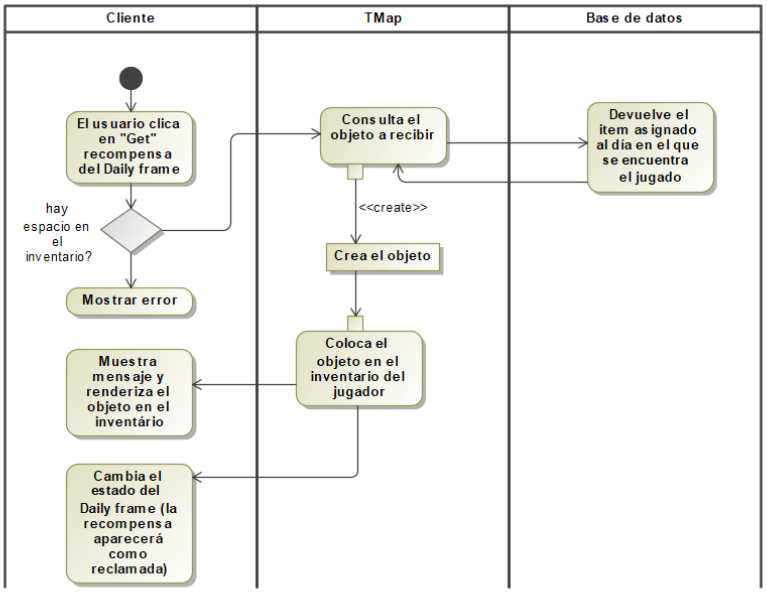

Figura 22. Diagrama de acciones de cómo se obtiene una recompensa del Daily frame

4.2. Recamara Battle

En la actualidad existen pocos servidores de 4Story capaces de construir sus propios modos de juego, pues normalmente esto conlleva una gran inversión de tiempo, a la vez que supone un sistema complejo y largo de probar.

4Legacy se encuentra con las características suficientes para poder crear su propio modo de juego, puesto que cuenta con herramientas de todo tipo que permiten diseñar frames, nuevos mapas, agregar misiones y muchas otras cuestiones de gran utilidad. A modo de reto, quise cumplir con el objetivo de ser uno de los programadores con el gusto de lograr dicho objetivo, y para ello surge la idea de la “Recamara Battle”.

Este modo de juego es de carácter sencillo, pues busca ser una modalidad rápida y divertida dentro de 4Legacy que pueda ejecutarse con una alta frecuencia a lo largo del día (existen muchos modos de juego que son de larga duración y, por lo tanto, se realizan en horarios punta para reunir más cantidad de jugadores).

Recamara Battle consiste en la guerra entre dos bandos, los cuales deben de estar lo máximo de equilibrados posible. Cada jugador partícipe en la guerra cuenta con 3 vidas, las cuales una vez agotadas conllevan en la expulsión del estadio de dicho jugador. En caso de la aniquilación absoluta de uno de los bandos, la batalla queda ganada para el bando

opuesto, mientras que si ninguno de los dos logra eliminar al resto de jugadores, el resultado viene determinado por el bando que mayor cantidad de jugadores vivos contenga.

A continuación se detallan los pasos a seguir para implementar este modo de juego dentro del TMap. Es importante recordar que los modos de juego suelen estar diseñados en un servicio independiente, por lo tanto, a pesar de que la Recamara Battle se encuentre en una versión prácticamente final dentro del TMap, resultaría mucho más útil, segura y eficiente si se creara un servicio (esta tarea está en proceso, pero se detalla más adelante sobre el diseño que se está siguiendo y debería de seguirse para finalizar).

En cuestión de componentes del cliente, la Recamara Battle requiere de la necesidad de un pequeño botón para permitir la inscripción a dicha batalla y de un frame el cual muestre el estado actual de la cola de la Recamara (indicando el tiempo restante y dando lugar a la cancelación de la cola).

Además, se añade un pequeño frame que aparecerá una vez dentro de la batalla para mostrar las vidas restantes al jugador.

Se describe, por lo tanto, en el siguiente fragmento de la documentación, los paquetes principales que se necesitan para la creación de la Recamara Battle dentro del TMap seguidos de sus correspondientes descripciones.

RecamaraStart: Este paquete se sitúa en el timer del TWorld, pues comprueba si la hora coincide con el horario indicado en la base de datos para iniciar la batalla. En caso de coincidir, este paquete se lanza al TMap, de forma que este pueda recorrer un listado con todos los jugadores del juego para ver quienes son los que cumplen con ciertas condiciones para así enviarles una notificación de que la batalla comenzará en los próximos 10 minutos, y, por lo tanto, disponen de ese tiempo para unirse.

La petición es enviada al cliente, quien, activa el frame para apuntarse a la batalla al jugador que le llega la petición.

RecamaraTimeUpdate: El frame correspondiente a la situación de la cola para la Recamara es gestionado por el cliente (cuestión de diseño decidida para no enviar un paquete a cada jugador partícipe durante cada tick del TMap), de forma que es posible según la velocidad de ejecución de este que quede atrasado. Este paquete es llamado con origen cliente destino TMap si es que el jugador utiliza de algún portal en el juego. Por otra parte, se llama con origen TMap destino cliente cada 2 minutos tras la hora de preparación de la guerra.

La utilidad que aporta este paquete es actualizar el contador de tiempo del frame para que quede ajustado al tiempo real y restante en el cual se ejecutará el comienzo de la Recamara Battle.

RecamaraBegin: Es invocado desde el mismo timer y una vez han pasado 10 minutos desde el lanzamiento del aviso. Este paquete es el encargado de inicializar la batalla, por lo tanto, es enviado al TMap.

La primera tarea que realiza es enviar un paquete **RecamaraCloseMenu** a todos los jugadores, pues este paquete cierra todos los frames referentes a la Recamara, pues la batalla va a comenzar y ya no es necesario visualizar la situación de la cola ni dar lugar a inscripción. Este paquete es recibido por el cliente.

A continuación, RecamaraBegin cuenta la cantidad de jugadores de cada reino que se han unido a la batalla, y realiza un cálculo sencillo para saber cuántos debe de situar en cada uno de ellos para que esté equilibrada (esto permite que jugadores de un reino puedan

pelear por el reino contrario para así no quedarse sin espacio en la batalla y además generar una diferencia máxima de 1 jugador extra en uno de los bandos).

Seguidamente, se produce dicho equilibrio, de forma que los jugadores que primero se encuentran en la cola serán los que tengan más prioridad por luchar con su reino, y, por lo contrario, los últimos de la cola serán más expuestos a modificaciones. Para producir este cambio de reino se necesita modificar el atributo bCountry que contienen los jugadores que se desean cambiar, pero también es necesario la llamada a un paquete **RecamaraChangeCountry**, el cual se enviará al cliente de todos los partícipes en la batalla para que rendericen a un jugador en el reino por el cual va a luchar (sin este paquete, los jugadores de su reino original no podrían detectarlo como un enemigo, de forma que no podrían atacarle). El cliente se encarga, por lo tanto, de cambiar dicho atributo también.

El paquete además realiza llamada a funciones ya existentes en el TMap que permiten eliminar los grupos creados (para asegurarnos que si alguno de los miembros cambia de reino no estarán en un mismo grupo). E inicializar monturas, compañeros, tiendas, etc. Una vez todo es inicializado, los jugadores son teletransportados al estadio de la Recamara Battle.

El RecamaraBegin, continúa su ejecución enviando un paquete al cliente de cada jugador para que renderice el frame que corresponde a la Recamara y que indica al jugador de cuantas vidas dispone (en un inicio 3). Este paquete es la respuesta de RecamaraBegin para el cliente.

Es necesario, por lo tanto, la llamada a **RecamaraDieUpdate**, el cual también es llamado para cada jugador y sirve para actualizar en cliente la variable que contiene la cantidad de vidas que dispone (3).

Finalmente, se llama a la función que comprueba si la batalla ha terminado, esta comprobación es necesaria por si en algún caso hay un o ningún jugador apuntado, de forma que no habría enemigos y la batalla finalizaría con un empate o victoria por parte del único jugador disponible.

RecamaraPacificTime: En un inicio, los partícipes de la batalla no pueden atacar a los enemigos puesto que se encuentran en un mapa definido como pacífico. Esta cuestión de diseño es emprendida para dar a los jugadores un pequeño tiempo de preparación y creación de grupos. Este paquete, por lo tanto, gestionado por el timer del TWorld, envía peticiones al TMap para que avise a cada jugador durante los 10 segundos antes del comienzo de la guerra. Cuando se ha terminado el tiempo, se llama al paquete **RecamaraPacificTimeEnd**, este se dirige al TMap también para recorrer el listado de jugadores y poder enviar una petición al cliente. El cliente indicará que la región ha dejado de ser pacífica y, por lo tanto, pueden invocar habilidades hacia los enemigos (las cuales serán aceptadas por el servidor puesto que la batalla ha comenzado).

RecamaraTimeOut: Una vez transcurrida la duración máxima del evento, este paquete es invocado por el timer del TWorld. El TMap recogerá el paquete y realizará el recuento de jugadores vivos en cada reino para proclamar un ganador y avisar y recompensar a los integrantes. Además, se devolverá a los jugadores al mapa original y a su reino, además de cerrar el frame de las vidas de la Recamara Battle, gracias a los paquetes RecamaraChangeCountry y RecamaraCloseMenu introducidos anteriormente.

Además, existen otras funciones que interactúan a la hora de suceder la Recamara y que no vienen determinadas por timers.

RecamaraDie: Esta función es recogida dentro del método original del juego OnDie que se ejecuta cuando un jugador muere en el TMap. Es invocada en el caso que el jugador que muere se encuentre dentro del mapa de la Recamara Battle. Se particiona en dos peticiones, una que realiza sobre el mismo TMap para que avise a todos los jugadores de las vidas restantes del jugador que ha muerto (esto permite que el cliente de todos los partícipes de la batalla sepa cuántas vidas quedan a cada jugador y, por lo tanto, pueda renderizar un pequeño componente sobre cada jugador indicando la cantidad de vidas restantes). Este paquete llama a RecamaraDieUpdate al cliente para todo el listado de jugadores indicando la cantidad de vidas restantes del afectado.

Además, esta petición es reenviada al cliente para que actualice el frame del jugador afectado y renderice en la propia pantalla el frame de sus vidas actualizadas.

Finalmente, es necesario que el TMap devuelva al jugador a su estado original (región, mapa, etc.) y compruebe si tras la eliminación de dicho jugador la batalla ha finalizado.

RecamaraQueue: Paquete enviado por el cliente hacia el TMap una vez es pulsado el botón del componente que permite apuntarse a la batalla. El servidor se encarga simplemente de añadir el jugador al listado de partícipes.

RecamaraQueueDelete: Si el jugador interacciona con el frame que muestra el estado de la cola para poder salir de esta, se llama a dicho paquete. Este llega al TMap, quien busca si el jugador estaba en el listado de partícipes y lo elimina.

Si el usuario realiza logout del personaje en cualquier momento, es necesario que salga de la cola. De la misma forma, si la batalla está comenzada, se debe de expulsar al jugador de la batalla y comprobar si esta ha finalizado tras la expulsión del mismo jugador.

Existen otras pequeñas llamadas a estos paquetes, de la misma forma que se interactúa con alguno de ya existente para añadir pequeñas utilidades tales como enviar recompensas a cada jugador, enviar mensajes generales, etc.

A continuación se muestra un diagrama sobre el proceso de colas de la Recamara:

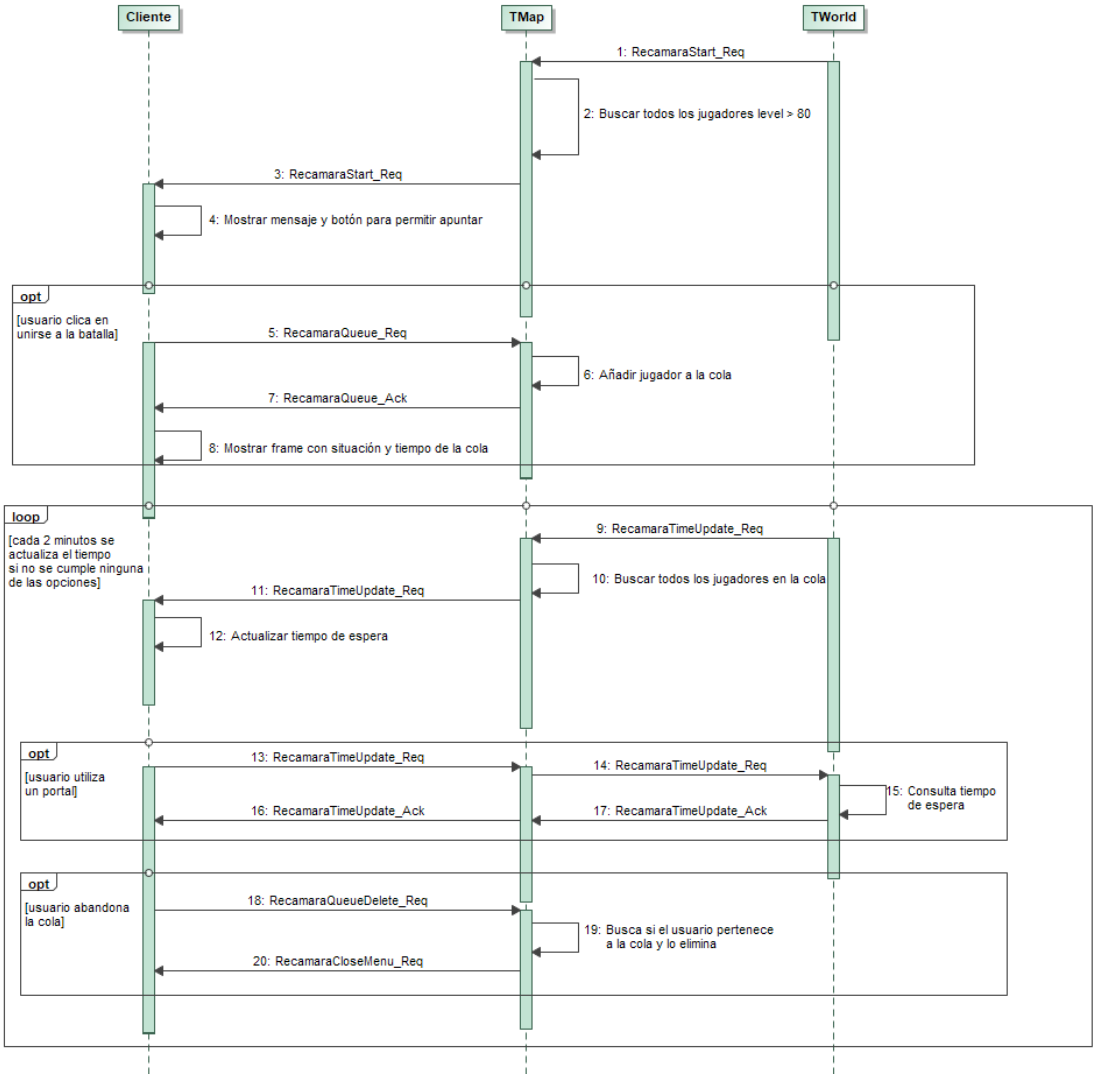

Figura 23. Diagrama de secuencia de las peticiones para iniciar la Recamara Battle

Este es el proceso que se lleva a cabo a la hora de comenzar la batalla:

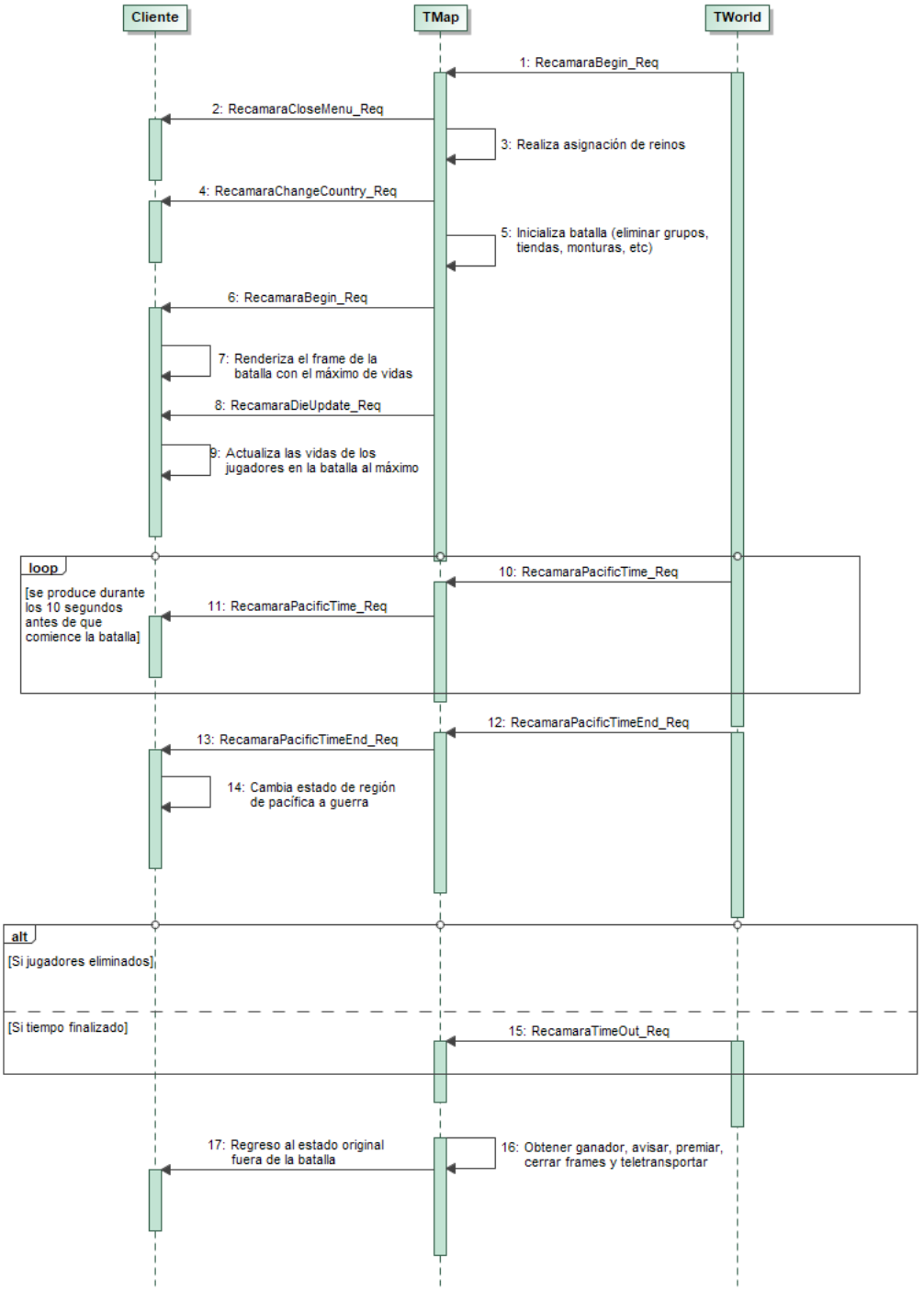

Figura 24. Diagrama de secuencia de las peticiones para comenzar la Recamara Battle

4.2.1. TRecamaraBattle Service

La implementación de la Recamara Battle como parte del TMap supone un alto riesgo, pues cualquier error, bug o vulnerabilidad supondría una consecuencia muy grande para el juego. Es por eso que la necesidad de crear un servicio independiente es algo de gran importancia en el caso de querer llevar este modo de juego a un escenario real.

El proceso de creación de un nuevo servicio se encuentra avanzado para el caso de la Recamara, es por eso que a continuación se detallan las cuestiones de diseño que se deben de seguir para lograr dicha meta.

Por parte del servidor, un nuevo servicio implica la importación en este de todo el código ya existente en el TMap. Una vez realizada esta tarea, es necesario leer y modificar todo el código que va a quedar inutilizado o debe de cambiar su comportamiento dentro del nuevo servicio. Para aplicar esta tarea se hace uso de la directiva #ifdef que proporciona C++. Por lo tanto, es necesario desactivar todos los paquetes y eventos que lanzan acciones que no se desean realizar.

Una vez se ha inutilizado todo el código que no se requiere dentro de la Recamara Battle, es hora de modificar el comportamiento de los paquetes que deben actuar de una forma distinta a la original.

Además, los modos de juego suelen definir sus propios inventarios, creando así un inventario por defecto que será el que se le proporcionará a cada jugador. De este modo, por lo tanto, los objetos de ambos TMap y TRecamaraBalle son independientes, proporcionando menor lugar a vulnerabilidades creadas a causa del traslado de los datos entre ambos servicios. Esta misma dinámica se suele seguir para todos los aspectos del juego, creando la mayor autonomía posible entre los elementos que intervienen en un modo de juego, de forma que toda la interacción y progreso que se realice en la batalla suele ser independiente y afectar lo mínimo posible al servidor corriente. Con tal de incentivar que los jugadores participen en esta batalla se ofrecen recompensas una vez terminadas la batalla, la cual cosa suele ser lo único que un jugador puede recibir de la conexión a dicha batalla.

La Recamara debe contener sus propias tablas en la base de datos con tal de guardar los jugadores que participan, la configuración general de esta (horas de ejecución, duración de la batalla, etc.) y los objetos que se proporcionan en los inventarios de los jugadores.

Por último, muchos de los frames, menús y botones que proporciona el cliente deben de ser inhabilitados de la misma forma que el servidor. El Daily frame por ejemplo, debe de considerar el caso en el que el jugador se encuentre en una batalla. En el caso contrario, un jugador podría reclamar una recompensa estando en una batalla y hacer uso de esta, además, perdería el objeto una vez fuera trasladado al TMap.

5. Juego de pruebas

La corrupción de un archivo .tcd puede suponer la manipulación de datos de los cuales no se tiene constancia, del mismo modo que puede generar pérdidas de información. Es por eso que es necesario realizar un juego de pruebas que permita verificar el uso correcto de cada una de las aplicaciones.

Para las aplicaciones más simples, las pruebas a realizar han verificado los siguientes aspectos:

- La información que la aplicación recibe a la hora de cargar un archivo es coherente y presenta el formato y el tipo esperado.
- Los archivos corruptos, incompletos o inválidos son rechazados por la aplicación.
- Los componentes de la aplicación muestran de forma correcta el atributo que representan.
- El archivo generado no manipula ningún bit en el caso de no realizar modificaciones.
- El archivo generado intercambia los bytes correspondientes a un único elemento en el caso de la modificación de este.
- El archivo generado mantiene la integridad respecto de si se crea o se elimina un elemento.
- El archivo generado permite ser leído por la aplicación generando los mismos resultados.
- Los botones de copiar no modifican ninguna instancia.
- Una vez se pega un elemento, este es modificado debidamente, de forma que todos los atributos quedan idénticamente clonados.
- Cuando pega un elemento y se modifica, no cambia los datos de la instancia de la cual se ha realizado la copia.
- El botón de guardado de un único elemento permite conservar los datos en el caso de seleccionar un nuevo elemento y volver atrás.
- El botón de guardado devuelve errores en el caso de la aparición de campos inválidos o incompletos.
- Cada uno de los elementos visuales actualizan la información y los datos que permiten visualizar si estos son modificados (en el caso del listbox, se actualiza si es que un elemento cambia su atributo índice, si se crea o elimina un nuevo elemento).
- Los componentes que permiten realizar búsquedas en los listbox devuelven únicamente las instancias esperadas.
- Una vez realizada una búsqueda sobre el listbox y vuelta a la perspectiva inicial (en la que se muestran todos los elementos), se muestran correctamente todas las instancias existentes e iniciales.
- Si se modifica una instancia realizando una búsqueda en el listbox, no se modifica ninguna otra instancia una vez vuelto el estado original del listado.

En cada una de estas pruebas se ha verificado la integridad de los datos tanto dentro de la aplicación (de forma que todos los elementos y componentes visuales contengan el resultado esperado) como en el archivo resultante, el cual se ha comparado con el original para ver si las diferencias entre ambos eran únicamente las deseadas.

En la siguiente prueba se muestra como tras haber cargado el archivo TItem.tcd (correspondiente a los objetos del juego e interpretado por la aplicación TItem) y modificado un único valor (el correspondiente al nombre del primer item), se mantiene la integridad de los datos.

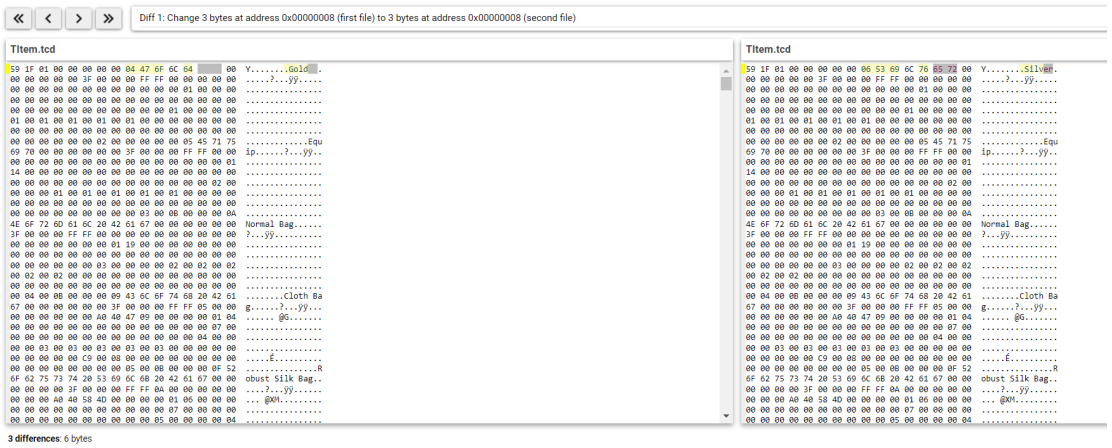

Figura 25. Muestra de la comparación del contenido binario tras la prueba

Las aplicaciones que necesitan conexión a base de datos han sido implementadas y probadas en bases de datos locales, de forma que se pueda verificar la integridad de los datos sin causar molestias ni corrupción de información oficial.

Estas, además de someterse a las pruebas anteriormente mencionadas, han supuesto la necesidad de comprobar bajo la realización de cada acción que hace uso de la base de datos que:

- La tabla afectada mantiene sus datos de forma idéntica al estado inicial si es que solo se desea realizar una consulta.
- Tras una acción que realiza una eliminación de una fila, no se genera la eliminación ni modificación de cualquier otro dato.
- Tras la creación de una nueva fila, los atributos reciben el valor esperado.
- Si se añade una nueva columna, la id corresponde a la más pequeña de entre todas las disponibles.
- A la hora de actualizar una fila no se modifican elementos no deseados.
- Solo se realizan cambios en las tablas deseadas, de forma que las tablas en las cuales no se desea interacción mantengan sus datos en el estado original.

Hasta el día de hoy todas estas pruebas han resultado eficientes y suficientes para las aplicaciones, pero… Los problemas surgen a la hora de realizar juegos de pruebas dentro del juego.

5.1. ¿Cómo se realizan los juegos de pruebas en 4Legacy?

En un código tan extenso, es frecuente que mediante la modificación de una acción en código suponga la ruptura o alteración del comportamiento de otras. Pues para asegurarnos que la implementación y diseño de nuevo código es correcto, se hace uso de 3 entornos diferentes.

El primer entorno, o entorno de desarrollo, se ha desempeñado en la mayoría de los casos de forma local. En este caso, como yo era el encargado de realizar el mantenimiento del código, compilaba y ejecutaba el proyecto de forma local. Para disponer de pruebas en conjunto se ha utilizado la apertura de puertos para la conexión de personal autorizado.

En los casos en los cuales algún otro compañero ha participado en el mantenimiento del código, se ha transladado el entorno de desarrollo a un VPS (Virtual Private Server) de bajo coste.

Este entorno hace referencia al estado del código en el cual se desarrolla, comprueba y verifica el código en una primera instancia. En este, las pruebas suelen estar realizadas por el propio programador, pues se comprueba básicamente que el código interactúe de la forma esperada (es decir, que cumpla con las necesidades que se esperan). En el caso de satisfacer la necesidad que se desea, el código es trasladado al siguiente entorno.

El segundo entorno, correspondiente al entorno de pruebas o testing, se encuentra hosteado en un VPS al cual se permite acceso a los administradores, testers y contactos estrechos o autorizados. En este se produce uso constante y cotidiano, pues los usuarios cumplen con la necesidad de probar el código actualizado. Para dicha tarea, se cuenta con un documento el cual proporciona a los testers información sobre las novedades que se han implementado y se detalla la función o el resultado esperado para cada una de ellas.

Si todo va bien, durante dos semanas el código se mantiene en fase de pruebas bajo constantes interacciones y valoraciones por parte del personal. Una vez verificado el código, está listo para ser actualizado en el servidor oficial.

En caso de invalidez o rechazo de algún aspecto implementado, el código vuelve al estado de desarrollo hasta generar una nueva propuesta y diseñar una solución estable.

El entorno final, el cual equivale al entorno de producción o de release, es el servidor oficial de 4Legacy y en el cual participan todos los jugadores. En este entorno no se puede realizar una actualización constante, puesto que esto produce normalmente el rechazo por parte de los jugadores. Además, supone una buena costumbre el hecho de acumular novedades para lanzar un parche o versión con muchas más noticias, pues los jugadores se sienten motivados tras leer que el servidor progresa y los desarrolladores trabajan de forma constante.

Para lanzar actualizaciones en este entorno es necesario planificar y organizar el contenido a aplicar. Por otro lado, es importante programar una fecha y avisar a los usuarios, la cual, se intenta lograr en un horario de baja disponibilidad o uso por parte de los usuarios finales.

6. Lanzamiento oficial

El día 8 de abril de 2022 se lanzó el juego de forma oficial a los usuarios. Durante los días anteriores se lanzaron pequeñas campañas publicitarias, al igual que se realizó el alquiler y montaje del servidor oficial en un VPS mucho más costoso y preparado.

El lanzamiento supuso en éxito en cuanto a la cantidad de usuarios que quisieron acceder al juego, en el Discord (plataforma que usa 4Legacy de forma oficial para mantener contacto cercano con la comunidad) llegaron a interactuar grupos de hasta 1000 personas. En el momento del lanzamiento, alrededor de 450 usuarios estuvieron a la espera para poder acceder a 4Legacy.

A la hora de realizar el despliegue del launcher y, por lo tanto, la apertura del servidor, comenzaron los primeros problemas:

- El servidor que hosteaba el launcher (el cual no dotaba de un hardware completo y avanzado para solventar un gran número de peticiones a la vez) no pudo soportar la carga de 500 usuarios realizando peticiones. Esto era fácilmente evitable, pues si los usuarios al realizar la primera descarga del juego hubieran dotado de la última versión de este con todos los archivos actualizados, no hubieran tenido que actualizar todos en el mismo instante. Pero, en cambio, el equipo cometió el primer pequeño fallo a la hora de crear una versión inicial, la cual además contenía archivos de grandes dimensiones (teniendo en cuenta que un gran flujo de peticiones llegaría al launcher).

  El launcher podría haber sido transportado a otro vps durante los días siguientes, tras observar la cantidad de gente interesada en el juego (pero el error ya estaba cometido, y no quedaba otra opción en ese instante que esperar a que la saturación del launcher se viera reducida).

- Más de 250 usuarios consiguieron acceder al juego de forma simultánea (flujo, el cual por nuestra parte era inconsiderable), creando una pequeña saturación del servicio de login. De forma inmediata se tomaron pequeñas medidas tales como evitar las tareas más duraderas como realizar escrituras por consola.
- Los primeros ataques al servidor llegaron, pues en 4Story es totalmente común que la comunidad intente lanzar ataques de todo tipo, incluso contratar a expertos, para hacer un uso maligno de servidores nuevos o competentes.
- Algunas de las misiones del camino principal a seguir no se encontraban preparadas para muchos usuarios, causando cuellos de botella y grandes esperas.

Generalmente, los primeros días fueron difíciles, ninguno de nosotros creía poder haber llegado a tantos usuarios, pues hasta pocos días antes del lanzamiento oficial no eran muchos más de 200 en total los usuarios que conocían de la existencia de 4Legacy.

El problema del launcher se solucionó, pues una vez realizado el despliegue, los usuarios demandaban las actualizaciones en diferentes horas y no todos de golpe.

Por otra parte, muchos usuarios decidieron no confiar en 4Legacy y marcharon a lo largo de la primera semana, de forma que el servidor consiguió mostrarse más estable cada vez.

Los primeros 15 días, fueron más de 15.000€ los recaudados por 4Legacy mediante las ventas de objetos y contenido dentro del juego.

7. Conclusión

A forma de conclusión, este proyecto ha supuesto para mí un gran foco de aprendizaje, a la vez que una experiencia inolvidable.

Gracias a 4Legacy he tenido la oportunidad de conocer a nuevas personas y formar una amistad con ellas (2 años trabajando y manteniendo conversaciones de forma constante tanto de forma oral como escrita, dan de mucho tiempo para conocer a alguien).

Por otro lado, 4Legacy ha supuesto para mí un soporte y un plus de aprendizaje a todo el conocimiento recibido a lo largo de mi carrera. En muchas ocasiones y asignaturas, he comprendido el temario con mucha más facilidad, además de haber gozado con una experiencia de primera mano sobre el funcionamiento sobre algunos de los conceptos introducidos (son ejemplos el caso de la asignatura de Sistemas Distribuidos, en la cual se explicaba el funcionamiento cliente-servidor, o la asignatura de Sistemas de Comercio Electrónico, pues yo ya había conseguido emprender y vivir uno de forma propia).

A día de hoy, aparte del aprendizaje y el entretenimiento, este proyecto supone para mí un pequeño añadido a mi curriculum, pues muchas empresas buscan a programadores Junior con mínimo un año de experiencia, y a veces la participación en este proyecto consigue llamar la atención de todos aquellos quienes leen mi curriculum. El hecho de haber obtenido responsabilidades y haber trabajado en un equipo ha supuesto en varias ocasiones el interés de algunos headhunter.

En cuanto a los resultados del proyecto, por mi parte, he de decir que han supuesto un mal sabor de boca, pues soy una persona bastante ambiciosa y de carácter negativo, y el hecho de haber estado tan cerca de conseguir juntar y gustar a tanta gente y haber perdido dicha oportunidad me hace sentir un poco insatisfecho. Por otra parte, es cierto que a nivel externo, lo que hemos logrado entre 4 jóvenes inexpertos ha sido algo maravilloso y admirable, pues prefiero pensarlo desde esta perspectiva.

4Legacy ha sido siempre para mí un proyecto y un hobby, pues yo no he realizado ninguna inversión más allá del tiempo en este videojuego. Es cierto que mis compañeros han sufrido una mayor presión en cuanto a la rentabilidad de este proyecto, pero he tratado de luchar y ayudar en todo lo posible a que todos rentabilicen su inversión.

El hecho de haber dispuesto de un personal limitado, al igual que no haber cumplido con experiencia anterior, ha supuesto para mí, al igual que para mis compañeros, la vivencia de unos meses difíciles de afrontar, sobre todo en los periodos anteriores y posteriores al lanzamiento del juego.

A estos factores se le añade la personalidad que tengo, la cual me hace pelear e implicarme al máximo por mis objetivos, y concluyen en la pérdida de motivación por mi parte de cara a 4Legacy. Los últimos meses han supuesto para mí un gran estrés, pues fuera del tiempo invertido en programar, no conseguía desconectar del proyecto a pesar de intentar practicar otros hobbies, estudiar o pasar tiempo con la familia. Además, la mayoría de estos días seguía un horario inestable, el cual resultaba en acostarme a las 6 de la madrugada, pues durante la noche era cuando se realizaban las actualizaciones y disponíamos de más tranquilidad.

Finalmente, el agotamiento y la pérdida de la motivación por parte de otros compañeros ha desencadenado en pequeños problemas con el equipo de carácter económico y organizativo.

Como es usual, los usuarios han perdido la motivación y confianza en el servidor a lo largo del tiempo, y a pesar de trabajar día y noche en mejorar la calidad de este y resolver las peticiones de los usuarios, cada acción negativa ha supuesto mucho más perjudicial que todo lo beneficiosos que han supuesto todos los aciertos.

A lo largo del tiempo, también los ataques de otras figuras destacadas en el ámbito de 4Story, al igual que el descubrimiento y manipulación de posibles vulnerabilidades, han generado la inestabilidad del sistema. A esto se le suma la corrupción y el mal uso de los privilegios por parte de algunos usuarios que se habían seleccionado como staff del servidor para ayudar, gestionar y dar soporte a la comunidad.

Como conclusión, se puede creer que todos los aspectos negativos que se han vivido desde el lanzamiento de 4Legacy han venido definidos por falta de personal y/o experiencia, pero lo cierto es que desde mi perspectiva el sistema de testing que seguíamos no ha resultado ser eficiente tampoco.

A pesar de mostrar 3 entornos de pruebas claramente definidos, muchas veces es difícil encontrar usuarios que vayan a probar el código y vayan a hacer un uso frecuente y exhaustivo de cada implementación añadida sin ofrecer un incentivo significativo. Además, a veces no ha sido cuestión de falta de pruebas, también ha faltado gente para simular ciertas pruebas. En ocasiones se han creado caídas del servidor, sobrecargas de servicios o retrasos como resultados a cuello de botella de algunos métodos.

Mi valoración es que este proyecto ha supuesto para mí una gran ventaja en muchos aspectos, con lo cual no volvería atrás en el tiempo para impedir que esto sucediera, pero, por otro lado, considero que no se ha llevado a cabo en el momento ni de la forma precisa de mi vida, pues mejor dejaría estar proyectos así de completos y con tanta necesidad de inversión para empresas y me limitaría a participar en proyectos con otras características. Un código mucho menos extenso, anticuado y manipulado daría lugar a una solución más estable, a la vez que posiblemente pueda suponer una inversión mucho más rentable en tiempo y beneficio.

8. De cara al futuro

En la actualidad, aún mantengo contacto con los amigos más cercanos por parte de 4Legacy, pero ya no me encuentro dispuesto como un miembro del equipo más. He tomado una pausa a lo largo de este verano del proyecto, pues 4Legacy ha permanecido cerrado durante todo este periodo para decidir arreglar muchos de sus errores cometidos, diseñar nuevos conceptos y tratar de comenzar de cero.

Mi posición se mantiene en que no voy a volver a retomar responsabilidades dentro del proyecto, puesto que no consigo gestionar mi tiempo y mente para llevarlo adelante sin estrés y como un hobby. Además, mis ganancias dentro de 4Legacy considero que no son suficientes ni están bien gestionadas.

Por lo tanto, como resultado a ambos problemas, desde ahora en adelante yo daré soporte a 4Legacy realizando el diseño de las aplicaciones y herramientas que se me asignen a cambio de un precio a ajustar por cada una de ellas.

Es común que siempre quiera implicarme en más de lo que me pertenece, puesto que no deja de ser algo que me apasiona, y, por lo tanto, de forma independiente es posible que de cara a un futuro consiga acabar logrando mi objetivo de crear un modo de juego a través de un servicio. Para ello, consideraré si el estado actual de la Recamara Battle se puede retomar.

9. Anexos

Para finalizar con esta documentación, se dispone de este breve anexo en el cual se va a detallar sobre el diseño de la base de datos de 4Legacy, pues ciertas aplicaciones han hecho uso de esta, de la misma forma que ha sido necesario su comprensión para la creación de nuevas tablas y métodos útiles para el código.

Por otra parte, se describe de forma simple el funcionamiento y diseño de algunas aplicaciones no mencionadas que aportan pequeñas modificaciones y características respecto a las definidas anteriormente.

Por último, se presentan algunas cuestiones de código más específicas, las cuales han sido parte de los nuevos rasgos implementados para 4Legacy.

9.1. Base de datos

Del mismo modo que el servidor, la base de datos se encuentra fragmentada en dos bloques, llamados TGAME y TGLOBAL

Se hace uso del TGAME para guardar información, la cual será específica para un servidor en concreto. Por lo general, el TMap es quien destina sus llamadas a este bloque.

Son ejemplos de tablas del TGAME:

- TQUESTTABLE: Tabla que almacena todas las misiones de las que dispone un personaje, del mismo modo que almacena todas las que han quedado completadas.
- TCHARTABLE: Contiene el nombre de todos los personajes creados en el juego, además, relaciona a cada personaje con la cuenta a la que pertenece, y muchos otros atributos como la raza, sexo, nivel, etc.
- TITEMCHART: Almacena todos los objetos existentes en el juego.

En las tablas del TGAME se diferencian dos sufijos: TABLE y CHART. La diferencia entre ambos es clara, pues las tablas que acaban por CHART hacen referencia al conjunto de un elemento que existe en el juego (por ejemplo, TMONSTERCHART almacena los monstruos existentes, TQUESTCHART dispone de la variedad de misiones que el servidor ofrece a los jugadores). Las tablas con el sufijo TABLE acostumbran a ser más extensas, pues contienen para cada elemento las instancias de las que dispone o ha dispuesto cada jugador (por ejemplo, TITEMTABLE almacena todos los objetos que tienen todos los jugadores del juego, TSKILLTABLE contiene todas las habilidades que cada personaje ha aprendido en el juego, así como el progreso que ha realizado con cada una).

Por otro lado, el TGLOBAL almacena información general, la cual resulta útil en diferentes servidores.

El TGLOBAL acostumbra a contener tablas que representan las cuentas de cada usuario. En consecuencia, todas las tablas que interactúen con las cuentas y no con los personajes, quedan asignadas también a este bloque.

Son ejemplos de tablas del TGLOBAL:

- IPBLACKLIST: Tabla que contiene la IP de algunos de los usuarios a los cuales no se les desea dar acceso al juego.
- TCOUPONTABLE: Contiene los diferentes cupones activos en el juego, a los cuales se le permite el uso al usuario en la web para obtener recompensas (se utilizan principalmente para estrategias de marketing, tales como promover y ayudar a los nuevos jugadores).

- TACCOUNT_PW: Dispone de las cuentas de todos los usuarios, conjuntamente con su contraseña encriptada y otros atributos (como por ejemplo la última vez que una cuenta ha recibido una recompensa diaria).

Hasta el momento, las modificaciones más extensas a la base de datos, por mi parte han sido la creación de métodos, útiles para realizar llamadas desde el código e interactuar con la mayoría de características implementadas.

9.2. TSFX

Esta aplicación se creó con la finalidad de poder permutar algunos de los efectos especiales, del mismo modo que para permitir investigar y estudiar sobre algunos de los efectos que 4Story contiene por defecto, pero los cuales permanecen inactivos.

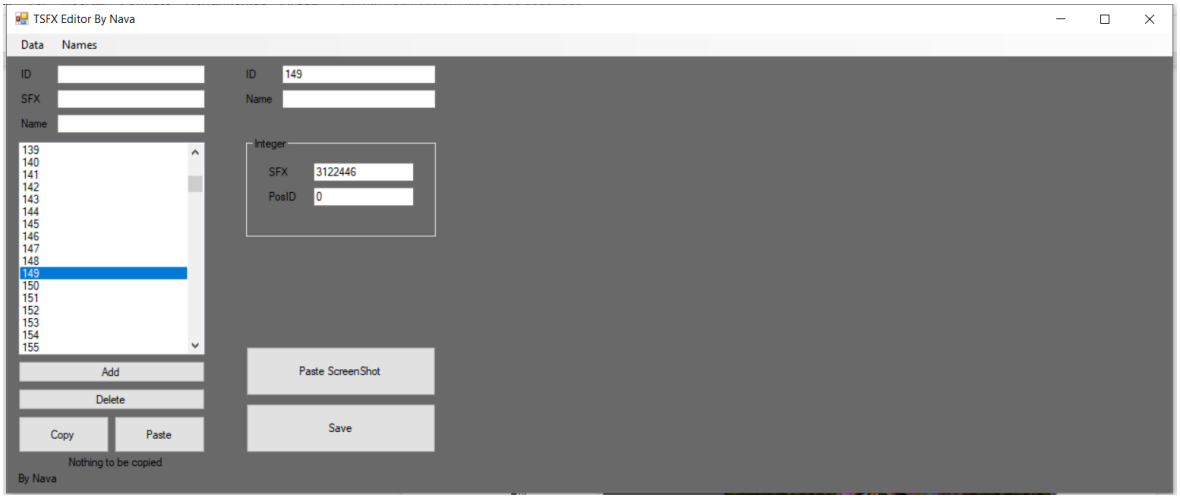

Figura 26. Aspecto visual de la aplicación TSFX

4Story contiene con una gran cantidad de información, la cual, por ahora, no queda documentada de ningún modo, es por eso que en muchas ocasiones es importante proporcionar en las aplicaciones nuevas características que faciliten la diferenciación y la comprensión de sus datos.

En el caso de esta aplicación, cada efecto especial recibe una id, pero resulta poco práctico diferenciar cada SFX mediante una id, pues es insostenible el hecho de recordar el aspecto de cada efecto.

Para ello, algunas de las aplicaciones contienen alguna información extra:

- Names: Permiten asignar un nombre a cada instancia.
- Img: Se da la posibilidad de almacenar una fotografía la cual permita mostrar el aspecto de la instancia.

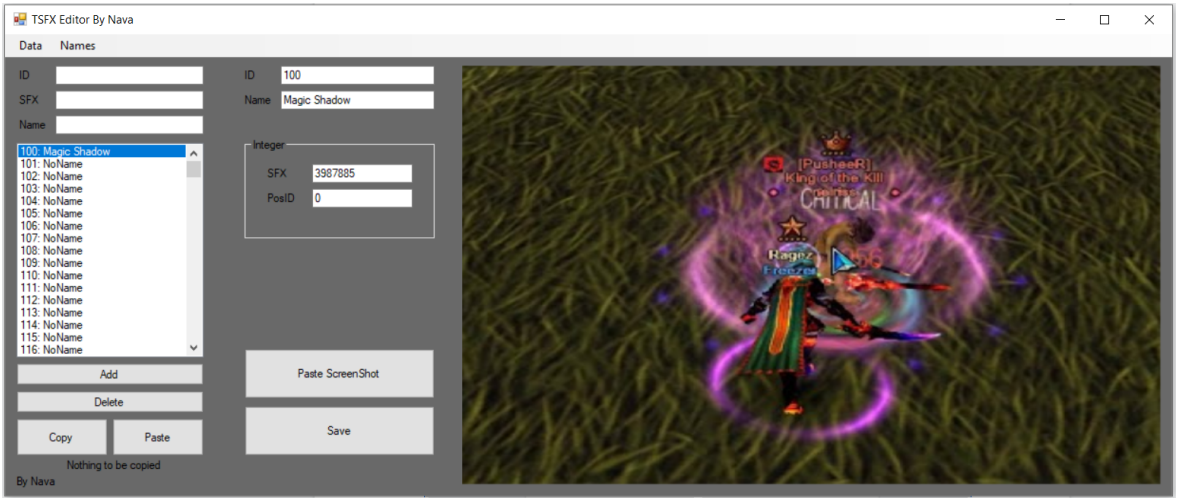

Figura 27. Aspecto visual de la aplicación TSFX con nombres y imágenes

Las aplicaciones que disponen de estos dos nuevos atributos, los gestionan de forma que:

- Permiten la creación de un fichero de texto inicial (para aquellos usuarios que interactúen con la aplicación por primera vez). Este fichero asigna por defecto NoName a todas las instancias.
- Crean una carpeta para almacenar las imágenes en el mismo lugar donde se encuentra la aplicación.
- Muestran flexibilidad frente a si el usuario dispone y quiere utilizar los nombres y las imágenes, tanto como si no se desean cargar.
- No es necesario que cada instancia reciba un nombre o una imagen, pues ambos serán de valor nulo si no son definidos.

A modo de implementación, la clase TSFX, la cual contiene todos los atributos de cada efecto especial, dispone de un atributo extra llamado Name. Este atributo no es cargado del .tcd, pues solo se iniciará si es que se realiza la carga del archivo TSFXNames.tcd.

La carga de cada imagen sucede a la hora de seleccionar una instancia en el listbox, pues la aplicación se encarga de buscar en la carpeta SFX la imagen con el nombre <id>.png.

Ambos imagen y nombre permiten ser cambiados desde dentro de la aplicación, pues esta permite pegar la captura de pantalla, la cual el usuario debe de haber realizado préviamente, del mismo modo que se permite cambiar el nombre y guardar el archivo TSFXNames.tcd una vez terminado su uso.

9.3. Nuevas habilidades

4Legacy ha sido el primer servidor en ofrecer a los jugadores un balance de clases constante, justo y divertido, pues es común en 4Story que predomine el uso de arqueros o magos, mientras que el resto de clases acostumbran a ser secundarias.

Para el diseño de este balance de clases se han necesitado meses de trabajo, pues muchas de las animaciones y características ofrecían errores o se presentaban inmutables. Dispuestos a crear un sistema de habilidades prácticamente nuevo, emprendí la opción de poder diseñar nuevas mecánicas y habilidades al juego. Para ello, como es lógico, lo primero era estudiar como estas funcionan.

Nuevas mecánicas han sido agregadas, tales como crear habilidades que denieguen la curación del enemigo, evitar ser ralentizado, volver atrás en el tiempo o realizar heridas graves.

Cada mecánica requiere de la modificación de diferentes funciones ya existentes en el código, pero en este caso nos centramos en la implementación de las heridas graves. Heridas graves hace referencia a una mecánica la cual, de forma momentánea, reduce las curaciones que el jugador puede recibir en un 60%.

Para la creación de esta mecánica es necesario la implementación de un STATUS_TYPE. Pues cada habilidad presenta un tipo, el cual hace referencia a la función que cumple la habilidad (por ejemplo, las habilidades de tipo SDT_ITEM permiten al usuario interactuar con objetos). La habilidad a crear es de tipo SDT_STATUS, puesto que aplica un estado al objetivo. El estado que se aplicará se asigna a cada habilidad, en este caso, como se deseaba crear una nueva mecánica, hay que crear un STATUS TYPE, el cual he llamado SDT_STATUS_GRIEVOUSWOUNDS (existen gran cantidad de tipos de estado, como por ejemplo, SDT_STATUS_DIE, el cual indica que el jugador está muerto y, por lo tanto, cambiará su comportamiento).

La siguiente tarea realizada es acudir a todas las funciones que se desean modificar para indicar que las curaciones apliquen un 60% del efecto a los jugadores con heridas graves.

Para analizar si un jugador contiene heridas graves, se dispone de un listado de MaintainSkill, los cuales hacen referencia a las habilidades que se mantienen, es decir, que no realizan un efecto instantáneo, sino que permanecen en el jugador durante un tiempo.

Es tan sencillo como recorrer este listado consultando si alguna de las habilidades que el jugador tiene aplicadas es de tipo SDT_STATUS y su tipo de estado es SDT_STATUS_GRIEVOUSWOUNDS.

Alguno de los ejemplos de funciones modificadas para aplicar esta mecánica son las funciones de usar un objeto (para evitar que las pociones curen su valor máximo) o la de usar una habilidad (para evitar que los personajes que realizan curaciones o transfusiones de vida, las apliquen al 100%).