Anexo B. Manual de uso de CrisolBuilder.

Índice.

B.1. Qué comprende CrisolBuilder.

B.2. Tipos de archivos.

B.3. Conceptos sobre el formato de los datos.
B.3.1. El tratamiento de los archivos musicales.
B.3.2. El tratamiento de los archivos gráficos en cuanto a las zonas invisibles
B.3.3. Las especificaciones para los archivos gráficos.
B.3.4. El color y la transparencia.
B.3.5. El concepto de quad.
B.3.6. El archivo TileRegionMask.

B.4. El archivo de Reglas.
B.4.1. Sección [GameController].
B.4.2. Sección [FloorTypes].
B.4.3. Sección [CriatureTypes].
B.4.4. Sección [CriatureClasses].
B.4.5. Sección [CriatureGenres].
B.4.6. Sección [CriatureExtendedAttributes].
B.4.7. Sección [CriatureEquipmentSlots].
B.4.8. Sección [CriatureHabilities].
B.4.9. Sección [CriatureSymptoms].
B.4.10. Sección [Combat].
B.4.11. Sección [SceneObjectTypes].
B.4.12. Sección [ItemTypes].
B.4.13. Sección [ItemGlobalAttributes].
B.4.14. Sección [WallTypes].
B.4.15. Sección [MinDistances].

B.5. El archivo de Plantillas de animación.
B.5.1. Conceptos generales sobre las animaciones.
B.5.2. Definición de un perfil de animación.

B.6. Archivos de perfiles asociados a entidades.
B.6.1. El archivo de perfiles asociado a suelos.
B.6.2. El archivo de perfiles asociado a techos.
B.6.3. El archivo de perfiles asociado a paredes.
B.6.3. El archivo de perfiles asociado a ítems.
B.6.4. El archivo de perfiles asociado a objetos de escenario.
B.6.5. El archivo de perfiles asociado a criaturas no jugador (npc's).

B.7. El archivo de especificación de áreas.
B.7.1. Conceptos generales sobre la organización de un archivo de área.
B.7.2. La sección [AreaInfo].
B.7.3. La sección [RoomProfiles].
B.7.4. Las secciones destinadas al diseño de las celdas.

B.8. El archivo de perfiles para la creación de un jugador.
B.8.1. Sección [Profiles].
B.8.2. Sección [Portraits].
B.8.3. Sección [Common].

B.9. El archivo de configuración de interfaces.
B.9.1. Conceptos generales.
B.9.2. Sección [Cursors].
B.9.3. Sección [Selectors].
B.9.4. Sección [AdviceDialog].
B.9.5. Sección [QuestionDialog].

B.9.6. Sección [TextFileReader].
B.9.7. Sección [TextOptionsSelector].
B.9.8. Sección [MainMenu].
B.9.9. Sección [LoaderSavedFileManager].
B.9.10. Sección [SaverSavedFileManager].
B.9.11. Sección [PlayerCreator].
B.9.12. Sección [GameLoading].
B.9.13. Sección [ConversationWindow].
B.9.14. Sección [EntityInteract].
B.9.15. Sección [InventorySelector].
B.9.16. Sección [TradeItemsManager].
B.9.17. Sección [CombatientInfo].
B.9.18. Sección [MainGameInterfaz].
B.9.19. Sección [GameMenu].
B.9.20. Sección [PlayerProfileWindow].

B.10. El archivo de definiciones GFX.
B.10.1. Sección [FXDefs].
B.10.2. Secciones de perfiles de GFX.

B.11. El archivo de plantillas de presentación.

B.12. El archivo de texto estático.
B.12.1. Sección [CriatureTypes].
B.12.2. Sección [CriatureClasses].
B.12.3. Sección [CriatureGenres].
B.12.4. Sección [CriatureHabilities].
B.12.5. Sección [CriatureSymptoms].
B.12.6. Sección [CriatureEquipmentSlots].
B.12.7. Sección [CriatureAttributes].
B.12.8. Sección [Interfaces].
B.12.9. Sección [Messages].

B.13. El archivo de scripts globales.
B.13.1. Sección [WorldEvents].
B.13.2. Sección [GameEvents].

B.14. El archivo “CrisolEngine.ini”.
B.14.1. Sección [SysVar].
B.14.2. Sección [Timer].
B.14.3. Sección [FontSystem].
B.14.4. Sección [GraphicSystem].
B.14.5. Sección [AudioSystem].
B.14.6. Sección [GameDataBase].

B.15. Utilización de CrisolBuilder.
B.15.1. El archivo “CrisolBuilder.ini”.
B.15.2. Tipos de mensajes.

B.1. Qué comprende CrisolBuilder.

Este documento abarca la utilización de la utilidad *CrisolBuilder* y explica cómo debén construirse los archivos de definición de contenido. *CrisolBuilder* está asociado a *Crisol* como soporte para poder construir los archivos binarios (“`.cbb`”) relacionados con las animaciones, áreas y perfiles de entidades así como para chequear los archivos de texto (“`.cbt`”) destinados a los perfiles de presentación, configuración de interfaces, reglas, perfiles del jugador, texto estático y de definiciones de GFX.

Así pues *CrisolBuilder* será una utilidad muy importante pues necesariamente se deberá de ejecutar al menos una vez para poder chequear los archivos de definición de datos (los archivos con extensión “`.cbt`” de los que se hablará más adelante) y, en algunos casos, generar, a partir de ellos, sus archivos binarios equivalentes. Antes de poder explicar cómo funciona *CrisolBuilder*, se deberá de comentar de forma exhaustiva cómo están constituidos los archivos de datos con el fin del que el usuario pueda configurar las animaciones y demás elementos para su juego. Además, también se explicarán los conceptos básicos en torno al formato de los datos gráficos y musicales que *Crisol* será capaz de manejar.

B.2. Tipos de archivos.

Los primeros, los de extensión ”.cbt”, serán archivos de texto mientras que los segundos, los archivos con extensión ”.cbb”, serán archivos binarios y se generarán a partir de sus homólogos “.cbt”. *Crisol* trabajará tanto con los archivos de texto como con los binarios. El elegir un formato u otro será por motivos de velocidad de procesamiento y posibilidad de ser modificados durante la ejecución.

Los archivos “.cbt” se podrán construir con cualquier editor de archivos de texto. Todos estos ficheros servirán para definir datos, indicar configuraciones, especificar parámetros, etc. Todos estos ficheros tendrán el mismo formato general:

```
[Seccion]
 Datos
End
```

Formato general de las secciones de un fichero” .cbt”.

Pudiéndose repetir una o más veces dicha construcción. En [Seccion] se incluirá siempre el nombre de aquel apartado donde el motor irá a buscar unos datos en concreto sabiendo que una vez encontrado, lo único que deberá de hacer será leer todo el contenido de la sección (parte denominada Datos) hasta llegar a End, que hará de límite de la sección. En el interior de la sección la información siempre se establecerá a partir de asignaciones, es decir, siempre tendremos construcciones del tipo:

```
Variable = Valor
```

Donde Variable vendrá impuesta por el motor (y comentada siempre en este documento) y Valor será una información que el diseñador proporcionará.

Los valores que se podrán asignar podrán ser de texto o bien numéricos. Nunca se podrá dejar una variable sin valor. Los únicos valores en formato texto que tendrán un significado numérico serán true y false, donde true equivaldrá a un valor verdadero, valor 1, y false a un valor falso, valor 0. Los valores true y false siempre estarán asociados a aquellas variables que acaben en la palabra “Flag”.

Todas las secciones asociadas a un archivo “.cbt“ comentadas en este documento, siempre deberán de estar presentes aunque no se definan datos en ellas. De no ser así se producirán errores en el procesamiento de la información.

En el caso de los archivos de perfiles (los cuales se verán más adelante), las secciones serán creadas por el propio usuario ya que la idea será que dichas secciones definan perfiles. Por ejemplo, en el archivo de perfiles de criaturas cada sección definirá un perfil (un tipo) de criatura. El formato de las secciones será entonces así: [Profile<Nombre>] donde Nombre será la denominación que el diseñador quiera dar al perfil. Estas denominaciones serán identificadores. Un identificador será una cadena de texto sin espacios, sin acentos y sin el carácter ‘ñ’.

Siempre que se hable de que una variable tiene un valor por defecto, se querrá indicar que en caso de no poner esa variable se entenderá que se ha asociado el valor indicado por defecto.

Dentro de los archivos “.cbt” se permitirá incluir información que no será tenida en cuenta, es decir, se podrán incluir comentarios. Los comentarios estarán organizados en líneas, de tal modo que poniendo el carácter ‘;’ al comienzo de una línea se logrará la misma sea ignorada.

Las variables se podrán escribir tanto en mayúsculas como en minúsculas estando también permitido ponerlas en cualquier orden dentro de la sección a la que pertenecen. Sin embargo, se recomendará seguir el orden que se dé en este documento para que la búsqueda de las mismas sea lo más rápida posible. Esto mismo será aplicable al trabajo con las secciones.

B.3. Conceptos sobre el formato de los datos.

Antes de entrar en detalles sobre los diferentes archivos "`.cbt`" y la forma de especificar el contenido, se deberá de tener claro algunos conceptos básicos sobre el tipo de archivos gráficos, musicales y de sonido digitalizado que soportará el motor. También se comentarán algunos convenios y conceptos que será necesario saber para poder entender las explicaciones sobre la definición de datos que más adelante se darán.

B.3.1. El tratamiento de los archivos musicales.

Crisol trabajará con dos tipos de archivos musicales: los archivos MIDI y los archivos WAV. En ambos casos, los archivos no podrán hallarse empaquetados (no podrán incluirse en el archivo "`GameData.pak`" a través de la utilidad *CPAKTool*. Para más información consultar la documentación asociada a esta utilidad).

B.3.2. El tratamiento de los archivos gráficos en cuanto a las zonas invisibles

El motor trabajará única y exclusivamente con archivos gráficos de extensión tga. La razón de utilizar estos archivos es que podrán tener asociado un canal denominado alpha que para el motor proporcionará la información necesaria para distinguir las zonas visibles de las que no lo sean dentro del gráfico. Todo archivo tga que tenga un canal alpha habilitado, tendrá la posibilidad de definir, en el mismo, zonas de la imagen que se verán de color blanco y otras que se verán de color negro. El motor nunca dibujará las zonas de color negro. Normalmente, en lo que no es el canal alpha sino el gráfico de verdad, se asociará un color RGB como el 255,0,255 para resaltar la porción del mismo que será invisible. Naturalmente, si el gráfico tga en cuestión no posee canal alpha, todo el gráfico será dibujado. Si por el contrario existiera canal alpha y en el mismo el color fuera todo negro, nada sería dibujado en pantalla. La herramienta gráfica Adobe Photoshop permite trabajar fácilmente con los gráficos de formato tga y editar el canal alpha, pero cualquier otra herramienta edición gráfica gratuita como el Paint Shop Pro o Gimp servirá.

A continuación se va a mostrar un ejemplo de un gráfico con canal alpha. En primer lugar se mostrará el gráfico tal cual, siendo la zona rosada la porción de gráfico que en el canal alpha se verá de color negro (no será dibujada):

Imagen de un archivo tga

Como se puede ver en la imagen, el gráfico podría ser el del cursor asociado al ratón. La zona rosa representaría la porción del mismo que no querremos que se pinte en pantalla. Con poner esta zona de color rosa no lograremos nuestro objetivo, deberemos de crear un nuevo canal en el gráfico y toda la zona rosa pintarla de negro mientras que la zona destinada al gráfico propiamente dicho deberemos rellenarla de blanco. Como ya se ha comentado, realizar estas operaciones con una herramienta como el Adobe Photoshop u otras gratuitas será relativamente sencillo. He aquí el aspecto que tendrá el canal alpha asociado al gráfico:

Aspecto del canal alpha

Cuando el motor abra el gráfico para leerlo sabrá, consultando el canal alpha, que la zona de blanco será la porción del mismo que deberá de dibujar en pantalla y la zona de negro la que deberá de obviar.

B.3.3. Las especificaciones para los archivos gráficos.

Crisol es un motor que utiliza la librería DirectX para el trabajo con los gráficos. En concreto, se apoya en la interfaz Direct3D por lo que, además de requerir una tarjeta gráfica 3D, todos los gráficos que maneje serán tratados como texturas. Esto implicará una serie de restricciones en lo relativo a los gráficos y es que todos deberán de tener unas dimensiones que sean, a lo alto y a lo ancho, potencias de dos. En concreto, se esperará que todos tengan valores de 32 a 256, esto es, 32, 64, 96, 128, 160, 192, 224 y 256 pero teniendo siempre presente que lo más común será trabajar con las dimensiones de 32, 64, 128 y 256 ya que muchas tarjetas gráficas no soportarán dimensiones de 96, 160, 192 o 224.

En lo relativo a cómo suministrar los gráficos de gran tamaño que puedan ejercer de imágenes de fondo, únicamente se deberá de tener en cuenta que el motor los particionará siempre en texturas de 32x32 (el diseñador no se tendrá que preocupar en realizar partición alguna) y que, por lo tanto, con tal de tener unas dimensiones que sean potencia de dos, estará todo resuelto.

Otro detalle a tener en cuenta es que el formato de las texturas con las que trabajará el motor serán de 16 bits. En este sentido, ningún gráfico será aceptado si su bits por pixels es de 8 o 32. Esto será debido, de nuevo, a que se pretenderá conseguir el mayor grado de compatibilidad con las tarjetas gráficas existentes.

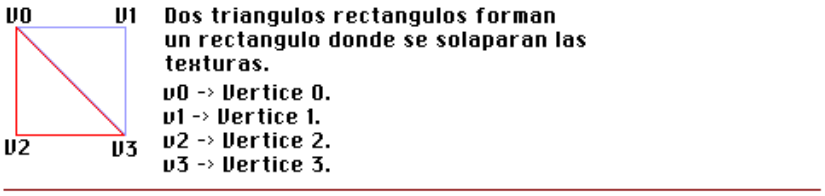
B.3.4. El color y la transparencia.

Cuando en el documento se hable de color, siempre nos referiremos a un formato en el que el valor del mismo se creará a partir de tres componentes, uno rojo (red), otro verde (green) y, por último, uno azul (blue) y lo denominaremos siempre color RGB. Los posibles valores de cada uno de los componentes podrán oscilar siempre entre 0 y 255. Siendo 0 el nivel más bajo del componente y 255 el más alto.

El valor de transparencia asociado a una imagen se denominará valor alpha (no confundir con el canal alpha de un archivo tga). Cuando se asocie un cierto grado de transparencia a una imagen, todo lo que se halle por debajo de ella, esto es, lo que se haya dibujado con anterioridad, se podrá ver en consonancia con el valor de transparencia que hayamos asociado a la imagen. Los valores posibles que podremos establecer oscilarán siempre entre 0 y 255, siendo 0 transparencia absoluta (no se verá el gráfico) y 255 solidez absoluta (no se verá nada de lo que haya por debajo).

B.3.5. El concepto de quad.

Todas las texturas serán asociadas, internamente, a unos objetos denominados quads. Dichos objetos no serán más que rectángulos (en realidad, dos triángulos rectángulos solapados). Será importante conocer que desde el punto de vista de la programación gráfica, será posible asociar valores de color y transparencia a cada uno de los 4 vértices de dicho rectángulo. En el único lugar en donde este factor será importante de conocer, será en el apartado donde se expliquen la creación de plantillas de GFX pues será posible asociar, en ese caso, valores de color y de transparencia de forma individualizada a cada uno de los vértices.



A este rectángulo, denominado quad, se le asociará una textura

B.3.6. El archivo TileRegionMask.

Para facilitar el trabajo del algoritmo de detección de las celdas seleccionadas por el cursor (en el área de juego), se utilizará un archivo gráfico especial denominado *TileRegionMask*. Sin este archivo *Crisol* no podrá funcionar. Siempbre deberá figurar en cualquier distribución del motor.

El aspecto de este archivo gráfico es el siguiente:

El archivo con la imagen TileRegionMask

Al diseñador no le tendrá que importar para nada dicho archivo, únicamente tendrá de indicar su nombre y la ruta de acceso en el fichero de inicialización de *Crisol* (que se verá más adelante). Para el desarrollador, decir que cada región de distinto color de la imagen supone incrementar o decrementar unos determinados valores a la posición del cursor para obtener, así, la localización exacta del mismo sobre el área isométrica.

B.4. El archivo de Reglas.

El archivo de reglas será el archivo de datos más importante ya que de él dependerá todo el funcionamiento lógico del motor y a partir de él se irán construyendo otros archivos de datos.

Antes de pasar a enumerar las distintas sus secciones, se deberá de tener en cuenta que en el archivo de reglas se definirán todas las informaciones relativas a las distintas entidades y al funcionamiento general del universo de juego. Cuando se definan las entidades, siempre será necesario especificar los tipos de entidades y, en algunos casos, las clases. Las clases podrán ser independientes de los tipos o bien dependientes de ellos. Cuando hablamos de especificar tipos y clases hablamos de definir cuantos tipos y clases existen de una determinada entidad y qué identificador tienen éstos. El identificador es el nombre que damos al tipo o a la clase y deberá de ser una cadena de texto sin espacios, sin acentos y sin carácter 'ñ'. Gracias a la definición de identificadores podremos referirnos a un conjunto de elementos sin usar un número. Y gracias a la especificación de estos, comunicamos también la cantidad de tipos, clases o atributos extendidos, por poner un ejemplo, que se van a utilizar.

Los identificadores de tipos y clases serán los más importantes pero también existirán identificadores relacionados con otros elementos de las reglas como los de géneros en las criaturas, los de habilidades, etc.

B.4.1. Sección [GameController].

En esta sección se especificarán los datos asociados al controlador del juego.

La variable `Time.SecondsPerMinute` servirá para especificar cuantos segundos posee un minuto de juego (en el mundo real un minuto son 60 segundos). Los valores permitidos estarán entre 15 y 60.

Ejemplo:

```
[GameController]
 Time.SecondsPerMinute = 20
End
```

B.4.2. Sección [FloorTypes].

Sección asociada a los suelos (`Floor`), en ella se especificarán los identificadores de los distintos tipos de suelos que existirán. La variable tendrá el formato `Type[x].ID` donde 'x' será el número de identificador, que comenzará en 0 siguiendo siempre un orden (0, 1, 2....). Como máximo se podrá llegar al valor 255. Las entidades `Floor` no tendrán clases, sólo tipos.

Ejemplo:

```
[FloorTypes]
 Type[0].ID = Tierra
 Type[1].ID = Agua
End
```

B.4.3. Sección [CriatureTypes].

En esta sección se especificarán los identificadores de los distintos tipos de criaturas que existirán. La variable tendrá el formato `Type[x].ID` donde 'x' será el número de identificador, que comenzará en 0 siguiendo siempre un orden (0, 1, 2....). Como máximo se podrá llegar al valor 255. Las criaturas también tendrán clases pero serán independientes de los tipos.

Ejemplo:

```
[CriatureTypes]
 Type[0].ID = Humano
 Type[1].ID = Hobbit
 Type[2].ID = Ogro
End
```

B.4.4. Sección [CriatureClasses].

En esta sección se especificarán los identificadores de los distintos tipos de clases asociadas a criaturas. La variable tendrá el formato `Class[x].ID` donde 'x' será el número de identificador, que comenzará en 0 siguiendo siempre un orden (0, 1, 2....). Como máximo se podrá llegar al valor 255.

Ejemplo:

```
[CriatureClasses]
 Class[0].ID = Paladin
 Class[1].ID = Guerrero
 Class[2].ID = Mago
 Class[3].ID = Ladron
 Class[4].ID = Explorador
End
```

B.4.5. Sección [CriatureGenres].

En esta sección se especificarán los identificadores de los dos únicos géneros posibles asociados a las criaturas. El formato será `Genre[x].ID` donde 'x' tendrá un valor igual a 0 o a 1. En cualquier caso, se deberán de especificar los dos identificadores. Por ejemplo:

Ejemplo:

```
[CriatureGenres]
 Genre[0].ID = Varon
 Genre[1].ID = Hembra
End
```

B.4.6. Sección [CriatureExtendedAttributes].

En esta sección se establecerán los identificadores de los atributos extendidos. El formato será `ExtendedAttribute[x].ID`, donde 'x' hará referencia al atributo extendido al que queremos asociar el identificador. El valor máximo será de 7 (lo que supondrá 8 atributos extendidos).

Ejemplo:

```
[CriatureExtendedAttributes]
 ExtendedAttribute[0].ID = Destreza
 ExtendedAttribute[1].ID = Fuerza
 ExtendedAttribute[2].ID = Suerte
End
```

B.4.7. Sección [CriatureEquipmentSlots].

En esta sección se determinarán los identificadores asociados a los slots de equipamiento. El número máximo de slots de equipamiento será de 8 siendo obligatorio definir los dos slots principales (que vienen a representar las manos). El formato de la variable será `EquipmentSlot[x].ID`, donde 'x' hará referencia al slot al que vamos a asociarle un identificador. Como ya se ha indicado, al menos deberá de existir la definición para x igual a 0 y para x igual a 1, sabiendo que el valor máximo será de 7.

Ejemplo:

```
[CriatureEquipmentSlots]
 EquipmentSlot[0].ID = ManoIzquierda
 EquipmentSlot[1].ID = ManoDerecha
 EquipmentSlot[2].ID = Cabeza
End
```

B.4.8. Sección [CriatureHabilities].

En esta sección se determinarán los identificadores de las habilidades posibles. Por cada habilidad se deberá de especificar las variable `Hability[x].ID` para el nombre del identificador, `Hability[x].UseFlag` por si es una habilidad de uso (`true`) o no (`false`) y

`Hability[x].InCombatUseCost` que servirá para indicar el coste de uso de la habilidad cuando se quiera utilizar en combate. Esta variable sólo será necesario de especificar cuando se haya indicado que la habilidad en cuestión se puede usar. Por defecto la variable `Hability[x].UseFlag` tendrá valor `false`, así que en principio todas las habilidades serán no usables. Así mismo, el valor por defecto de `Hability[x].InCombatUseCost` será de 0, esto es, no costará usarla en combate. El número máximo posible de habilidades asociados a una criatura será de 8, por lo que x podrá valer a lo sumo 7.

Ejemplo:

```
[CriatureHabilities]
 Hability[0].ID = Magia
 Hability[0].UseFlag = true
 Hability[0].InCombatUseCost = 1
End
```

B.4.9. Sección [CriatureSymptoms].

En esta sección se especificarán los identificadores de los síntomas de las criaturas. El formato de la variable será `Symptom[x].ID`, donde x podrá oscilar entre 0 y 15 pues el número máximo de síntomas posibles será de 16.

Ejemplo:

```
[CriatureSymptoms]
 Symptom[0].ID = Envenenado
 Symptom[1].ID = Cansado
 Symptom[2].ID = Hambriento
End
```

B.4.10. Sección [Combat].

En esta sección se establecerán todos los parámetros asociados al combate. En general, estará dividida en una parte en la que se especificará los puntos de coste asociados a ciertas acciones y otra en donde se indicará el parámetro con que ordenar a los combatientes que entran en combate.

Así pues, tenemos las variables `ActionPoints.Walk` que indica el coste de andar de un tile a otro que es adyacente, `ActionPoints.Equip`, que indica el coste de equipar un ítem, `ActionPoints.Unequip` que indica el coste de quitar un ítem que esté equipado y devolverlo al inventario, `ActionPoints.UseFromInventory` que indica el coste de usar un ítem que se halle en el inventario sobre otra entidad, `ActionPoints.Observe` que indica el coste de realizar la acción de observar, `ActionPoints.GetItem` que indica el coste de recoger un ítem que se halle sobre el suelo, `ActionPoints.Manipule` que indica el coste de manipular una entidad que sea una pared y un objeto de escenario no contenedor (el resto de entidades están prohibidas durante combate de ser manipuladas), `ActionPoints.UseHability` para indicar el coste de usar una habilidad que pueda ser usada y que se añadirá al coste de uso de la habilidad concreta, `ActionPoints.Attak` para el coste de realizar un ataque con las manos (en el caso de que se ataque con un arma, este valor se añadirá al coste propio de usar el arma en combate). Por defecto, los puntos de coste en combate serán de 0 en todos los casos.

El último valor a indicar en esta sección estará referido a establecer el orden en que se deberá de generar la lista de criaturas que ingresen en combate. Para ello, sólo habrá que asociar el identificador de atributo extendido pertinente a la variable `TurnListOrderBy`. En caso de no especificar nada (omitir la asignación de valor a esta variable) se entenderá que se quiere ordenar la lista por el simple orden de inclusión de las criaturas en combate. Cuando indicamos el identificador de un atributo extendido, siempre ocuparán las primeras posiciones en la lista de combatientes aquellas criaturas que posean el valor más elevado de ese atributo extendido (en cuanto a su valor temporal).

Ejemplo:

```
[Combat]
 ActionPoints.Walk = 1
 ActionPoints.Equip = 1
 ActionPoints.Unequip = 1
 ActionPoints.UseFromInventory = 1
 ActionPoints.Observe = 3
 ActionPoints.GetItem = 1
 ActionPoints.Manipule = 1
 ActionPoints.UseHability =  1
 ActionPoints.Attack = 1

 TurnListOrderBy = Destreza
End
```

B.4.11. Sección [SceneObjectTypes].

Esta sección estará destinada a los objetos de escenario. Las clases de los objetos de escenario estarán supeditadas a los tipos siempre. Por cada objeto de escenario tendremos `Type[x].ID` para establecer el identificador del tipo y `Type[x].Class[y].ID` donde se especificará la clase asociada al tipo x. El valor de y podrá oscilar entre 0 y 255 pero teniendo en cuenta que siempre estarán supeditadas al tipo x. Asociado al identificador de tipo x también se podrán especificar los identificadores de atributos locales de un objeto de escenario usando la variable `Type[x].LocalAttribute[z].ID`, donde z podrá oscilar entre 0 y 3 pues el número máximo de atributos locales es de 4. Por último, se podrá especificar si el tipo x actual pertenece a un objeto de escenario contenedor mediante `Type[x].ContainerFlag`. Por defecto su valor será de `false`, es decir, no será contenedor. El valor de x asociado al identificador de tipo oscilará entre 0 y 255.

Ejemplo:

```
[SceneObjectTypes]
 Type[0].ID = Baul
 Type[0].Class[0].ID = Normal
 Type[0].Class[1].ID = Magico
 Type[0].LocalAttribute[0].ID = ResistenciaGolpes
 Type[0].ContainerFlag = true

 Type[1].ID = Columna
 Type[1].Class[0].ID = Piedra
 Type[1].ContainerFlag = false

 Type[2].ID = Arbol
 Type[2].Class[0].ID = Normal
 Type[2].Class[1].ID = Medio
 Type[2].ContainerFlag = false
End
```

B.4.12. Sección [ItemTypes].

En esta sección se establecerán los identificadores de tipos de ítems existentes. Al igual que con los objetos de escenario, las clases estarán asociados a los tipos. Así, tendremos `Type[x].ID` para indicar el tipo de identificador actual, `Type[x].Class[y].ID` especificará el identificador de clase asociado al tipo actual y `Type[x].LocalAttribute[z].ID` especificará el identificador asociado al atributo local z. El número máximo de atributos locales permitidos será de 6 luego el valor máximo de z podrá ser de 5. Mediante la variable `Type[x].Slot[k].CanEquipFlag` se especificará si los ítems de tipo x podrán equiparse en el slot k, siendo k el identificador del slot interesado. Por defecto, los valores para todos los slots serán `false`, lo que quiere decir que en caso de no poner nada para un tipo de ítem, será imposible equiparlo. La última variable estará relacionada con el coste de usar los ítems en combate y se especificará con la variable `Type[x].InCombatUseCost`. Por defecto el valor será de 0 y sólo se tendrá en cuenta esta variable si el ítem puede equiparse en uno de los slots principales (ya que para poder atacar con un ítem, necesariamente éste deberá de estar equipado en uno de los slots principales).

Ejemplo:

```
[ItemTypes]
 Type[0].ID = Arma
 Type[0].Slot[ManoIzquierda].CanEquip = true
 Type[0].Slot[ManoDerecha].CanEquip = true
 Type[0].InCombatUseCost = 1
 Type[0].LocalAttribute[0].ID = Danyo
 Type[0].LocalAttribute[1].ID = NivelMinimo
 Type[0].Class[0].ID = EspadaFuego
 Type[0].Class[1].ID = MazoPequenyo
 Type[0].Class[2].ID = Cuchillo

 Type[1].ID = Ropa
 Type[1].Class[0].ID = Camisa
 Type[1].Slot[Vestido].CanEquip = true
 Type[1].LocalAttribute[0].ID = Defensa

 Type[2].ID = Comida
 Type[2].Class[0].ID = Filete
 Type[2].Class[1].ID = Queso
 Type[2].Class[2].ID = Champinyon
 Type[2].LocalAttribute[0].ID = ValorAlimenticio
End
```

B.4.13. Sección [ItemGlobalAttributes].

En esta sección se especificarán los identificadores de los atributos globales asociados a los ítems. Los atributos globales serán aquellos que todo ítem poseerá independientemente de su tipo (recordemos que los identificadores de atributos locales estarán asociados a un tipo de ítem únicamente). La variable para especificar los atributos globales será `GlobalAttribute[x].ID` donde x podrá oscilar entre 0 y 2, pues el número máximo de atributos globales será de 3.

Ejemplo:

```
[ItemGlobalAttributes]
 GlobalAttribute[0].ID = Peso
End
```

B.4.14. Sección [WallTypes].

En esta sección se especificarán los tipos de pared existentes. Al igual que las entidades de tipo `Floor`, las paredes no tendrán clases, sólo tipos. Las variables a especificar serán `Type[x].ID` para especificar el identificador de tipo y `Type[x].LocalAttribute[y].ID` para especificar los identificadores de atributos locales. El valor de y podrá oscilar entre 0 y 3 pues el número máximo de atributos locales será de 4.

Ejemplo:

```
WallTypes]
 Type[0].ID = ParedNormal
 Type[0].LocalAttribute[0].ID = ResistenciaGolpes
End
```

B.4.15. Sección [MinDistances].

Esta sección tiene como objeto especificar las reglas asociadas a las mínimas distancias a las que se deberán de encontrar las criaturas para que se puedan efectuar ciertas acciones. Cuando dichas distancias sean factibles se ejecutará la acción relacionadas y cuando no sean posibles, la criatura se acercará. Las variables en esta sección estarán muy basadas en los identificadores que se hayan podido ir dando a los ítems y habilidades. Así pues, las variables serán `Talk` para indicar la distancia mínima para hablar, `UseHability[y]`, donde y será el identificador de la habilidad usable ya especificada y `UseWeapon[z]`, donde z será el identificador del ítem en calidad de arma (es decir, el ítem deberça de poder equiparse en slot principal y ser usado en combate para golpear).

Los valores podrán oscilar entre 0 y 20 teniendo en cuenta que 0 significará tener a la entidad sobre la que se va a actuar en un tile adyacente.

Ejemplo:

```
[MinDistances]
 Talk = 6
 UseHability[Magia] = 0
 UseWeapon[Arma].Class[EspadaFuego] = 1
 UseWeapon[Arma].Class[MazoPequenyo] = 1
End
```

B.5. El archivo de Plantillas de animación.

Las animaciones serán parte fundamental de la definición de contenido del motor. Cuando hablemos de animación realmente estaremos hablando del sprite que representa una entidad, un botón, etc.

B.5.1. Conceptos generales sobre las animaciones.

Toda animación tendrá asociada dos apartados fundamentales que la definirán. Por un lado estarán sus valores de configuración y por otro los estados de animación. Los valores de configuración servirán para indicar el número de estados de animación y orientaciones, las dimensiones de los gráficos que compongan los frames de animación y los fps (frames per second o frames por segundo) a los que se ejecutarán los frames de un estado de animación. Los estados de animación tendrán sus propias parcelas de configuración que indicarán, para cada una de sus posibles orientaciones, los frames de animación que las representarán. Dichos frames podrán ir desde 1 a 255. Naturalmente, si hay un solo frame de animación no habrá realmente animación.

Los frames de una animación representarán los “gráficos” que compongan la animación. Dicho de otro modo, si tenemos una animación de una vela necesitaremos, como mínimo, dos gráficos diferentes o de lo contrario no tendremos sensación de movimiento en la llama de la vela. El motor lo que hace es alternar dichos gráficos para transmitir animación. El funcionamiento es igual que en una película de dibujos animados. He aquí un ejemplo de los frames que pueden comprender una animación asociada a un personaje que anda en dirección este:

Animación de un personaje andando de izquierda a derecha (orientación) con un total de 15 frames.

Para que el motor pueda realizar las animaciones, deberá de tener el valor de los fps asociado a los frames, es decir, se le deberá de suministrar el número de frames de la animación que deberán de ser visionados en un segundo. Si el valor fuera 0, el motor no cambiaría nunca de frame de animación y por lo tanto no habría ningún tipo de sensación de animación. Los fps también serán muy valiosos para poder regular la velocidad a la que se ejecuta una animación. Algunas veces nos interesará tener animaciones lentas (entre 2 y 7 frames por segundo), normales (entre 8 y 18 frames por segundo) y muy rápidas (24 frames por segundo o más). En cualquier caso, estos valores habrá que irlos probando y ver los resultados.

Los estados de animación representarán, como su propio nombre indica, estados. Estos estados siempre agruparán orientaciones. Es necesario usar el concepto de orientación porque el motor es isométrico y los personajes que se desplacen sobre el mismo lo podrán hacer en 8 direcciones distintas, siendo el motor responsable de cambiar dichas orientaciones de forma automática. Así pues, toda animación tendrá como mínimo una orientación. Las animaciones que tengan una sola orientación serán, normalmente, todas aquella animaciones de interfaz (botones, cursores, etc) y también entidades que no cambien nunca ni de posición ni de orientación (por ejemplo, un objeto de escenario que represente árbol). Por lo tanto, una animación tendrá, al menos, un estado de animación y un estado de animación tendrá, al menos una orientación. Ésta última, a su vez, tendrá, al menos, un frame.

Siempre hablaremos de frames asociados a un estado, pues las orientaciones serán un pequeño filtro organizativo de los frames. La verdadera jerarquía de orden vendrá definida siempre por los estados y todas las posibles configuraciones, que a continuación serán comentadas, siempre estarán asociadas al concepto estado.

Como norma general pensaremos que cuando se alcanza el último frame de animación asociada a un estado (independientemente de la orientación actual de dicho estado) se

empezará por la primera de nuevo. Sin embargo, será posible indicar que cuando se alcance el último frame de animación se haga un cambio a otro estado de animación. Esto resultará muy interesante. Por ejemplo, supongamos la animación de una puerta en la que hay dos estados de animación, en uno se ven los frames de la puerta abriéndose y en otro se ve el frame de la puerta completamente abierta. Si se abre la puerta se ejecutarán todos los frames de animación asociados a tal acto y cuando se alcance el último frame el motor cambiará automáticamente de estado de animación mostrando un único frame; el de la puerta abierta. Este concepto se llama estado de enlace y todo estado de animación lo poseerá.

Cuando un estado de animación se tiene así mismo como estado de enlace, decimos que el estado de animación es cíclico. En estos casos será posible asociar un tiempo de espera entre ciclos completos de animación. En otras palabras, si tenemos un estado de animación en el que al terminar de ejecutarse todos los frames se vuelve al principio, podremos indicar que cuando se alcance el ultimo frame se espere un tiempo determinado en el primero antes de volver a empezar el ciclo. Esto puede ser interesante si tenemos un árbol que tiene animación en sus hojas simulando que hay viento. Si ponemos que una vez que se muevan las hojas se espere un par de segundos habremos conseguido un efecto de viento racheado mucho mas realista que hacer que las hojas se muevan todo el rato.

Los frames asociados a un estado también podrán reproducirse en orden inverso al que sería de esperar. Es decir, la norma general será reproducir los frames de izquierda a derecha pues bien, también se podrá indicar que se reproduzcan de derecha a izquierda. Esto es de capital importancia para ahorrar en material gráfico. Imaginemos otra vez el ejemplo de la puerta abriéndose y completamente abierta. Si queremos pasar del estado abierta al estado cerrada, tendremos que volver a ejecutar los frames que simulaban la apertura pero a la inversa. Si no pudiéramos ejecutar los frames en orden inverso, sería necesario disponer otra vez de los mismos frames pero colocados a la inversa. Esto supondría tener otro archivo con igual contenido pero en orden inverso y un gasto de tiempo y espacio apreciable para realizar dicha modificación. Puede que en un ejemplo no se note, pero cuando sean varias docenas de casos estaremos ante un ahorro fundamental.

Como último apunte sobre el funcionamiento general de las animaciones, decir que todo estado de animación podrá tener asociado un sonido WAV. Tener asociado un sonido WAV supondrá que cuando se active dicho estado de animación (como siempre, dando completamente igual la orientación que a su vez se active) el sonido WAV se reproducirá. Las particularidades nacen ahora de cómo se reproducirá dicho archivo WAV pues dependiendo de si el estado de animación es cíclico o no, el modo variará. Si el estado de animación es cíclico, se podrá indicar que el sonido se reproduzca de forma contínua y que sólo cese cuando se cambie de estado de animación o bien una vez alcanzado el último frame por primera vez. Si el estado de enlace apunta a otro estado distinto, no se pasará de estado hasta que el sonido WAV haya finalizado si bien, se podrá indicar si una vez alcanzado el último frame se esperará en este antes de cambiar o bien si se ejecutarán los frames de forma cíclica saltando automáticamente una vez que haya finalizado de reproducirse el archivo WAV.

B.5.2. Definición de un perfil de animación.

El formato del archivo es el correspondiente a cualquier archivo de plantillas, es decir, las secciones tendrán el formato `[Profile<Nombre>]` donde `Nombre` identificará el nombre de la plantilla de animación que se esté definiendo.

En primer lugar vendrán los campos destinados a la configuración general de la animación. Estos campos serán obligatorios. Las variables `Configuration.NumStates` y `Configuration.NumOrientations` servirán para indicar el número de estados y orientaciones con las que cuenta una animación. Nótese que al menos deberán de valer 1 ya que como mínimo deberá de existir un estado con una animación. Más adelante se verá que para ciertas animaciones se pedirá un número de estados y orientaciones puntual. Con `Configuration.FramesWidth` y `Configuration.FramesHeight` indicaremos las dimensiones de los frames asociados a la animación y con `Configuration.FPS` los frames por segundo. Es muy importante tener en cuenta que, como se verá a continuación, se suministra un gráfico a una orientación y a partir de las dimensiones de los frames y del

número de éstas para esa orientación el motor extrae los frames que componen la animación. Las dimensiones de las animaciones deberán ser siempre múltiplos de 2 a partir de 32 y sin superar nunca el valor de 256. (ver el apartado *Conceptos sobre el formato de los datos* para tener una idea más exacta sobre las dimensiones posibles).

Una vez especificada la configuración general de la animación, se pasara a definir los estados de esta que, necesariamente, deberán de ser igual al valor de `Configuration.NumStates`. A partir de ahora siempre se comenzará con `State[x].` donde x valdrá entre 0 y el valor asociado a `Configuration.NumStates` menos uno. Esto es, si especificó que hay un solo estado de animación, x podrá valer únicamente 0. Lo primero que habrá que indicar para un estado serán los datos de configuración general que le afectan. Uno de esos datos será el estado de enlace, que se especificará con `State[x].LinkState` y cuyo valor deberá de ser un estado válido (recordemos, los mismos valores que podrá alcanzar x). Su valor por defecto será el del estado actual, por lo que si no se pone nada en dicho apartado todos los estados existentes serán cíclicos. A continuación se esperará leer el tiempo de espera entre ciclos de animación mediante la variable `State[x].TimeBetweenCycles`. Dicho valor se buscará sólo si el estado es cíclico. Su valor por defecto será de 0, es decir, que no se contemplará tiempo de espera. Mediante la variable `State[x].ReversibleModeFlag` indicaremos si el estado de animación es reversible, esto es, si en lugar de reproducirse los frames de izquierda a derecha se reproducen de derecha a izquierda. Por defecto valdrá `false` con lo que no será reversible. Si se desea indicar que hay un archivo WAV asociado que debe de reproducirse, se deberá de tener en cuenta dos variables. Por un lado `State[x].WAVFileName` que servirá para indicar el nombre del archivo WAV a reproducir (como es lógico, por defecto no tendrá valor para indicar que no se desea asociar un archivo WAV). En caso de indicarse un nombre, se pasará a leer la variable obligatoria `State[x].WAVFlag` que servirá para indicar el flag que especificará el modo de reproducción. Dicho flag estará supeditado a que se haya definido, o no, un estado cíclico. Si el estado es cíclico y vale `true`, el archivo WAV se reproducirá continuamente y si vale `false` se reproducirá hasta que finalice. Si el estado no es cíclico y vale `true` el archivo WAV se reproducirá hasta que finalice pero se esperará su finalización en el último frame. Si vale `false` también se reproducirá hasta que finalice, pero los frames seguirán alternándose sin realizar espera en el último.

Una vez especificada la configuración del estado de animación, vendrá la parte destinada a suministrar los frames a las orientaciones de dichos estados. Recordemos que al menos deberá de existir una orientación. El archivo gráfico donde estén contenidos todos los frames asociados a la orientación del estado actual se especificará mediante la variable `State[x].Orientation[y].FramesFileName` (nótese que y será un valor que oscile entre 0 y el número total de orientaciones que se especificó menos uno). Todos los frames deberán de estar dispuestos en horizontal y uno detrás de otro si bien, se permitirá que en un mismo gráfico puedan insertarse varias filas de frames diferentes. Esto resultará muy útil para minimizar el número de archivos gráficos con los frames. Con la variables `State[x].Orientation[y].CellStartXPos` y por el otro `State[x].Orientation[y].CellStartYPos`, podremos indicar la fila y la columna en donde comenzar a leer los frames. Nótese que lo que realmente hará el motor será multiplicar el valor asociado a `State[x].Orientation[y].CellStartXPos` por la anchura de un frame y el valor indicado en `State[x].Orientation[y].CellStartYPos` por su altura. Como por defecto dichos valores tendrán valor 0, siempre se comenzará, si no se indica nada, en la posición 0,0 del gráfico. Leer un frame no será otra cosa más que situarse en la zona del archivo donde comienzan a estar y leer bloques de datos cuyas dimensiones sean las especificadas en la zona de configuración. Se leerán tantos como número de frames se hayan indicado.

Ejemplo:

```
[Profile<ATUpButton>]
 ; Plantilla asociada a un posible boton
 ; Nota: se definen dos estados porque uno sera el que tenga el boton
 ; si estar selecionado y otro sera el que tenga estando seleccionado.
 Configuration.NumStates = 2
 Configuration.NumOrientations = 1
 Configuration.FramesWidth = 128
```

```
 Configuration.FramesHeight = 32
 Configuration.FPS = 0
 State[0].NumFrames = 1
 State[0].Orientation[0].FramesFileName = Graphics\Interfaz\UpButton.tga
 State[1].NumFrames = 1
 State[1].WAVFileName = WAV\Click.WAV
 State[1].WAVFlag = false
 State[1].Orientation[0].CellStartXPos = 1
 State[1].Orientation[0].FramesFileName = Graphics\Interfaz\UpButton.tga
End

[Profile<ATPortrait>]
 ; Plantilla asociada a un posible retrato
 Configuration.NumStates = 1
 Configuration.NumOrientations = 1
 Configuration.FramesWidth = 64
 Configuration.FramesHeight = 64
 Configuration.FPS = 0
 State[0].NumFrames = 1
 State[0].Orientation[0].FramesFileName = Graphics\Npc\Npc00_Cara.tga
End
```

B.6. Archivos de perfiles asociados a entidades.

Los archivos de perfiles asociados a entidades hacen referencia a aquellos archivos de datos que sirven para definir una entidad de tipo suelo, techo, pared, ítem, objeto de escenario o npc (criatura no jugador) de tal forma que cuando desde el archivo que define las áreas queramos situar un suelo, un ítem, una pared, etc, en un lugar determinado tan solo tengamos que poner el nombre del perfil. El motor irá entonces al archivo adecuado y leerá la configuración que asociamos al mismo.

Esta forma de trabajar definiendo perfiles tiene la misma ventaja que con las plantillas de animación. Imaginemos que en un escenario queremos colocar 10 espadas en distintos cofres. Si no tuviéramos un sistema basado en perfiles, por cada vez que quisiéramos colocar cada una de esas espadas tendríamos que definir una serie de valores de forma redundante. Gracias a este sistema, basta con indicar el nombre del perfil espada a colocar y el motor sabrá acudir, dentro del archivo de perfiles adecuado, a los datos de configuración.

Todos los archivos de perfiles estarán organizados siguiendo el convenio de que las secciones serán de la forma `[Profile<Nombre>]` donde el valor asociado a `Nombre` será el identificador que le daremos al perfil y el que especificaremos siempre que queramos usarlo. En todos los archivos de perfiles se utilizarán de forma intensiva los identificadores definidos en el archivo de reglas y en el de plantillas de animación, por lo que habrá que tenerlo en cuenta antes de proceder establecer los datos de estos archivos.

Todas las entidades, en su representación gráfica, deberán de tener una anchura de 64 pixel para cada frame. Esto es así porque el motor trabajará con un mapa isométrico en el que las baldosas que formen el suelo tendrán dicha anchura. La altura, sin embargo, podrá exceder en esas dimensiones y oscilar entre 32 y 256 siempre en múltiplos de dos. En el único caso en el que no se podrá variar bajo ningún concepto las dimensiones de los frames que componen la imagen será cuando se esté trabajando con los ítems. En este caso, las dimensiones siempre deberán de ser de 64 x 64.

Cuando se quiera construir objetos de escenario (esto sólo valdrá para objetos de escenario) de dimensiones muy grandes a lo ancho que no quepan en un tile, se podrá dividir el gráfico que forme dicho objeto de escenario en partes y crear perfiles de objetos de escenario para cada una de esas partes. El siguiente paso será el situarlas sobre el escenario de forma adyacente. Se deberá de tener en cuenta que en este tipo de casos, como norma general, se querrá desactivar el flag de interacción pues aunque nuestro objetivo sea que la posición adyacente de dichos objetos formen el objeto grande que queremos mostrar (por ejemplo, un gran arbol) el mótor desde el punto de vista lógico las considerará zonas distintas así que si activaramos el flag de interacción podriamos estar interactuando con una porción del objeto de escenario nada más.

Como ejemplo a lo anterior, supongamos que tenemos un árbol muy alto y delgado, que no sobrepasa los 64 pixeles de ancho pero que llega a medir una altura de 384 pixeles. La idea será crear un pefil para la parte baja y otro para la parte alta. La parte baja podrá ir desde la parte inferior del gráfico 256 pixel hacia arriba y la parte alta desde el pixel 256 a lo alto hasta el 384 (realmente, una dimensión de 128 pues 384 = 256 + 128, en ambos casos múltiplos de dos permitidos). Cuando se estudie el diseño de áreas, se verá que podremos colocar el objeto de escenario que representa la parte baja del árbol en un tile y en ese mismo tile, el objeto de escenario que representa la parte alta pero con una elevación tal que encaje con la terminación de la parte inferior, mostrando así el árbol de gran altura que queríamos.

Todos los archivos de perfiles, salvo los asociados a los techos, se usarán también para la asociación de los eventos scripts de las distintas entidades. Dependiendo del tipo de entidad, así será el conjunto de scripts que se podrá asociar.

B.6.1. El archivo de perfiles asociado a suelos.

El archivo de perfiles asociados a los suelos, `Floor`, están destinados a definir, en líneas generales, los perfiles de los tiles con los que se establecerá el suelo de las áreas. El aspecto

gráfico de los suelos serán romboidal, tal y como se explica en el documento de diseño, teniendo éstos, obligatoriamente, unas dimensiones de 64 pixeles de ancho y 32 de alto. He aquí una imagen de ejemplo:

Tile de 64x32

El tile tendrá dos partes fundamentales: la rosada y la que define el conteido del mismo (su aspecto). La rosada será obligatoria, pues el motor dibuja primero todo el suelo y una vez hecho esto, comienza a dibujar, volviendo a recorrer el área, los elementos que estén por encima. Si se ocupara la zona rosada habría incoherencias gráficas. Naturalmente, la zona rosada será transparente.

La primera variable hace referencia a la plantilla de animación y se establecerá con la variable `AnimTemplate`.

A continuación se esperará leer el identificador del tipo de entidad (que se habrá definido previamente en el archivo de reglas), dicho valor se leerá a través de la variable `Type`.

Los accesos a tile se especificarán a través de las variable `Orientation[x].ObstacleFlag` siendo x un valor igual a `North` para indicar obstaculización al norte, `NorthEast` para indicar si hay obstaculización al noreste, `East` para indicar si hay obstaculización al este, `SouthEast` para indicar si hay obstaculización al sudeste, `South` para indicar si hay obstaculización al sur, `SouthWest` para indicar si hay obstaculización al sudoeste, `West` para indicar si hay obstaculización al oeste y `NorthWest` para indicar si hay obstaculización al noroeste. Por defecto todos los valores asociados a estas variables serán `false` por lo que el acceso estará permitido.

Con lo explicado ya se habrá definido un perfil de suelo sin embargo, faltará explicar el apartado destinado a asociar un sonido WAV al tile cuando éste sea pisado por una criatura. Para ello, existirá una sección denominada `[WalkWAVSound]` en donde para cada tipo de `Floor` existente se podrá asociar un archivo WAV que será reproducido cuando se pise sobre él. Por defecto no habrá ningún sonido asociado y por lo tanto no se reproducirá nada al pisar sobre el mismo. Para indicar el sonido asociado a un tipo usaremos la variable `Floor[x]` donde x será el nombre del identificador de tipo que proceda (por ejemplo `Floor[Tierra]` = archivo WAV). Será muy importante tener en cuenta que esta sección deberá de existir aunque no haya contenido alguno definido.

Para especificar el comportamiento que deberá de tener el `Floor` al quedar fuera de pantalla (esto es, cuando el jugador no lo vea), se podrá escoger entre realizar una pausa o bien bajar el volumen al máximo del posible sonido asociado a su plantilla de animación. Por defecto se realizará siempre lo primero, esto es, se pausará. Al pausarse, se deberá de tener en cuenta que no se podrá variar manualmente la plantilla de animación (esto es, cambiar de estado u orientación). Si queremos indicar manualmente que queremos establecer un modo de pausa, usaremos la variable `NoVisibleInScreenMode.PauseFlag` con valor `true`, mientras que si lo que queremos es indicar que se siga con actividad, pero bajando únicamente el sonido, donde deberemos de poner `true` será en `NoVisibleInScreenMode.SilenceFlag`. En caso de poner `false` en ambas, se utilizará el modo por defecto, esto es, la pausa.

Los eventos scripts se especificarán con `ScriptEvent[x]` donde x será el identificador del script. Por defecto se entenderá que no habrá script asociado a un evento así que si no se establece nombre alguno asociado a un evento simplemente significará que no existirá script para dicho evento. A la hora de asociar un script a un evento se deberá de especificar el nombre del archivo con su extensión. Los valores de x para los eventos asociados a los tiles serán `OnSetIn`, para indicar que una entidad se ha colocado sobre el tile, y `OnSetOut` para indicar que una entidad se ha salido del tile.

Ejemplo:

```
[WalkWAVSound]
 Floor[Tierra] = WAV\Pasos.WAV
End

[Profile<SueloTierra>]
 Type = Tierra
 AnimTemplate = ATSueloTierra
 NoVisibleInScreenMode.PauseFlag = true
 ScriptEvent[OnSetIn] = Script_OnSetIn_TilePlesor.cst
End

[Profile<SueloElevado>]
 Type = Tierra
 Orientation[SouthEast].ObstacleFlag = true
 Orientation[South].ObstacleFlag = true
 AnimTemplate = ATSueloElevado
End
```

B.6.2. El archivo de perfiles asociado a techos.

El aspecto gráfico de un techo seguirá los mismos patrones que los explicados para las entidades que formen los suelos. El archivo de perfil será bastante sencillo ya que únicamente habrá que especificar la plantilla de animación a través de la variable `AnimTemplate`.

Para especificar el comportamiento que deberá de tener al quedar fuera de pantalla (esto es, cuando el jugador no lo vea), se podrá escoger entre realizar una pausa o bien bajar el volumen al máximo del posible sonido asociado a su plantilla de animación. Por defecto se realizará siempre lo primero, esto es, se pausará. Al pausarse, se deberá de tener en cuenta que no se podrá variar manualmente la plantilla de animación (esto es, cambiar de estado u orientación). Si queremos indicar manualmente que queremos establecer un modo de pausa, usaremos la variable `NoVisibleInScreenMode.PauseFlag` con valor `true`, mientras que si lo que queremos es indicar que se siga con actividad, pero bajando únicamente el sonido, donde deberemos de poner `true` será en `NoVisibleInScreenMode.SilenceFlag`. En caso de poner `false` en ambas, se utilizará el modo por defecto, esto es, la pausa.

Ejemplo:

```
[Profile<Chimenea>]
 AnimTemplate = ATChimenea
End
```

B.6.3. El archivo de perfiles asociado a paredes.

En este archivo se definirán los perfiles asociados a las entidades de tipo pared. Recordemos que estas entidades servirán tanto para representar paredes como conceptos como puertas. El primer valor que se esperará leer será el de plantilla de animación asociada a través de la variable `AnimTemplate`.

A continuación se esperará leer el nombre de la pared a través de la variable `Name`. Por defecto, una pared no tendrá nombre asociado (por lo que si es interactuable y se coloca el cursor sobre la misma no aparecerá nombre alguno).

La plantilla de animación asociada al retrato se esperará recibir a través de la variable `Portrait.AnimTemplate`. Por defecto no habrá plantilla de animación así que este campo estará vacío.

La variable `ShadowImage` servirá para especificar la imagen asociada a la sombra. A diferencia de lo que se podría pensar no se esperará recibir una plantilla de animación sino una imagen suelta. Por defecto no tendrá valor asociado, para indicar que no hay imagen asociada como sombra.

El tipo de la pared que se está definiendo se especificará en la variable `Type`. Este campo será obligatorio y deberá de corresponder con un identificador definido en el archivo de reglas.

La orientación de la pared será un dato importante para el calculo de caminos cuando una criatura se tenga que mover entre ellas. Para especificar este valor se utilizará la variable `Orientation` cuyo valor deberá de ser `SouthWest` si la orientación es sudoeste, `Front` si es frontal o `SouthEast` si es sudeste.

Mediante la variable `IsInteractuableFlag` podremos indicar si la pared es interactuable. Ser interactuable significa que podrá ser seleccionada por el cursor de interacción por parte del jugador y que éste podrá actuar sobre ella examinándola o usando un ítem, por poner ejemplos. Por defecto una pared no será interactuable, es decir, el valor por defecto para esta variable será de `false`.

Los últimos campos estarán destinados a establecer unos valores iniciales para los atributos locales de la pared. Estos valores se pondrán a través de la variable `LocalAttribute[x]` donde x es el identificador del atributo local (definido en el archivo de reglas) sobre el que se quiere actuar. Por ejemplo `LocalAttribute[ResistenciaGolpes] = 10`. Por defecto los valores asociados a los atributos locales serán de 0.

Para especificar el comportamiento que deberá de tener la entidad al quedar fuera de pantalla (esto es, cuando el jugador no lo vea), se podrá escoger entre realizar una pausa o bien bajar el volumen al máximo del posible sonido asociado a su plantilla de animación. Por defecto se realizará siempre lo primero, esto es, se pausará. Al pausarse, se deberá de tener en cuenta que no se podrá variar manualmente la plantilla de animación (esto es, cambiar de estado u orientación). Si queremos indicar manualmente que queremos establecer un modo de pausa, usaremos la variable `NoVisibleInScreenMode.PauseFlag` con valor `true`, mientras que si lo que queremos es indicar que se siga con actividad, pero bajando únicamente el sonido, donde deberemos de poner `true` será en `NoVisibleInScreenMode.SilenceFlag`. En caso de poner `false` en ambas, se utilizará el modo por defecto, esto es, la pausa.

Los eventos scripts se especificarán con `ScriptEvent[x]` donde x será el identificador del script. Por defecto se entenderá que no habrá script asociado a un evento así que si no se establece nombre alguno asociado a un evento simplemente significará que no existirá script para dicho evento. A la hora de asociar un script a un evento se deberá de especificar el nombre del archivo con su extensión. Los valores de x para los eventos asociados a las paredes serán `OnObserve`, para indicar que la entidad ha sido observada, `OnTalk`, para indicar que el jugador desa hablar con la entidad, `OnManipulate`, para indicar que una entidad desea manipular a la entidad, `OnEntityCreated`, para cuando la entidad sea creada y `OnEntityIdle`, para especificar el script Idle asociado a la entidad.

Ejemplo:

```
[Profile<ParedNormal_0>]
 AnimTemplate = ATWall00
 Type = ParedNormal
 Orientation = SouthEast
End

[Profile<PuertaHierro_0>]
 Type = PuertaHierro
 Orientation = Front
 AnimTemplate = ATDoor
 NoVisibleInScreenMode.SilenceFlag = true
 LocalAttribute[ResistenciaGolpes]  = 13
 ScriptEvent[OnObserve] = Script_OnObserve_PuertaPlesor.cst
End
```

B.6.3. El archivo de perfiles asociado a ítems.

Aquí se definirán los perfiles asociados a los ítems. Los ítems siempre se caracterizarán por ser elementos muy versátiles de cara a ser localizados tanto sobre el terreno de un área como

sobre contenedores, entiendo por contenedor un objeto de escenario que así lo sea (un cofre por ejemplo) o bien el inventario de una criatura (incluida el jugador).

Los ítems serán las únicas entidades cuyas dimensiones deberán de ser, obligatoriamente, de 64 x 64. Ni más ni menos.

La primera variable que esperamos recibir será el nombre de la plantilla de animación. Este valor se recogerá a través de la variable `AnimTemplate` y deberá de estar definida en el archivo de plantillas de animación.

A continuación se leerá el nombre del ítem. Será importante que los ítems tengan siempre un nombre asociado ya que aunque no es obligatorio, son entidades interactuables siempre. El nombre se recogerá a través de la variable `Name`.

El retrato de un ítem será definido en la variable `Portrait.AnimTemplate` y su valor por defecto será nulo, esto es, que un ítem (al igual que cualquier entidad salvo el jugador) no tendrá porque tener asociado un retrato. Se esperará leer, en caso de que se quiera asociar un retrato, el identificador de la plantilla de animación que lo representa.

La variable `ShadowImage` servirá para especificar la imagen asociada a la sombra a diferencia de lo que se podría pensar no se esperará recibir una plantilla de animación sino una imagen suelta. Por defecto no tendrá valor asociado para indicar que no hay imagen asociada como sombra.

A continuación se esperará leer el tipo y la clase asociada al ítem. Recordemos que en este tipo de entidades la clase está supeditada al tipo. Para leer el tipo se usará la variable `Type` y para leer la clase la variable `Class`. Estos valores serán siempre obligatorios.

Por último se podrán especificar los valores asociados a los atributos locales y globales del ítem. Por defecto su valor será de cero. En el caso de los atributos locales se especificará a través de la variable `LocalAttribute[x]` donde `x` es el nombre del identificador del atributo local definido en el archivo de reglas. Los valores iniciales de los atributos globales se especificarán a través de `GlobalAttribute[y]` donde y vuelve a tener que ser un nombre de identificador de atributo global definido para los ítems.

Para especificar el comportamiento que deberá de tener la entidad al quedar fuera de pantalla (esto es, cuando el jugador no lo vea), siempre y cuando el ítem no esté en propiedad de una entidad conteneora, se podrá escoger entre realizar una pausa o bien bajar el volumen al máximo del posible sonido asociado a su plantilla de animación. Por defecto se realizará siempre lo primero, esto es, se pausará. Al pausarse, se deberá de tener en cuenta que no se podrá variar manualmente la plantilla de animación (esto es, cambiar de estado u orientación). Si queremos indicar manualmente que queremos establecer un modo de pausa, usaremos la variable `NoVisibleInScreenMode.PauseFlag` con valor `true`, mientras que si lo que queremos es indicar que se siga con actividad, pero bajando únicamente el sonido, donde deberemos de poner `true` será en `NoVisibleInScreenMode.SilenceFlag`. En caso de poner `false` en ambas, se utilizará el modo por defecto, esto es, la pausa.

Los eventos scripts se especificarán con `ScriptEvent[x]` donde `x` será el identificador del script. Por defecto se entenderá que no habrá script asociado a un evento así que si no se establece nombre alguno asociado a un evento simplemente significará que no existirá script para dicho evento. A la hora de asociar un script a un evento se deberá de especificar el nombre del archivo con su extensión. Los valores de `x` para los eventos asociados a las paredes serán `OnObserve`, para indicar que la entidad ha sido observada, `OnTalk`, para indicar que el jugador desa hablar con la entidad, `OnManipulate`, para indicar que una entidad desea manipular a la entidad, `OnEntityCreated`, para cuando la entidad sea creada, `OnEntityIdle`, para especificar el script Idle asociado a la entidad, `OnGet`, para cuando la entidad sea recogida, `OnDrop`, para cuando la entidad sea soltada, `OnUseItem`, para cuando la entidad sea utilizada y `OnTradeItem`, para cuando la entidad sea intercambiada.

Ejemplo:

```
[Profile<Hueso_0>]
 Type = Hueso
 Class = Calavera
 Name = Calavera
 AnimTemplate = ATItemCalavera_0
 ScriptEvent[OnGet] = Script_OnGet_Calavera.cst
 ScriptEvent[OnObserve] = Script_OnObserve_Calavera.cst
 End

[Profile<EspadaFuego_0>]
 Type = Arma
 Class = EspadaFuego
 Name = Espada de fuego
 GlobalAttribute[Peso] = 15
 LocalAttribute[Danyo] = 50
 LocalAttribute[NivelMinimo] = 5
 AnimTemplate = ATEspada_Fuego
End

[Profile<MazoPequenyo_0>]
 Type = Arma
 Class = MazoPequenyo
 Name = Mazo pequeño
 GlobalAttributs[Peso] = 13
 LocalAttribute[Danyo] = 14
 LocalAttribute[NivelMinimo] = 1
 AnimTemplate = ATMazo_Pequenyo
 Portrait.AnimTemplate = ATPlayerMaleFace00
End
```

B.6.4. El archivo de perfiles asociado a objetos de escenario.

Los perfiles asociados a los objetos de escenario, tanto aquellos que puedan ser contenedores (por ejemplo un cofre) como aquellos que no lo sean (por ejemplo un árbol), serán definidos en este archivo.

La primera variable que esperamos recibir es el nombre de la plantilla de animación. Este valor se recogerá a través de la variable `AnimTemplate` y deberá de estar definida en el archivo de plantillas de animación.

El nombre del objeto de escenario se esperará recibir a través de la variable `Name`. Al igual que ocurre con las paredes, los objetos de escenario podrán ser o no interactuables, cuando éstos lo sean el poseer nombre será algo importante para indicar al usuario con solo poner el cursor sobre el objeto el nombre del mismo. Por defecto no habrá ningún nombre asociado.

El retrato del objeto de escenario será opcional, esto es, no se esperará recibir nada, y en caso de asociarlo se esperará que sea a través de la variable `Portrait.AnimTemplate`. Para el retrato se esperará que se asocie el nombre de un identificador de plantilla de animación. En el caso de los objetos de escenario que se hayan definido como contenedores en el archivo de reglas, el asociarles un retrato tiene especial importancia pues la ventana de interfaz que se activa cuando son examinados buscará dicho retrato para mostrarlo en pantalla y si ese objeto de escenario no tiene retrato alguno, esa posibilidad visual no existirá al no mostrarse nada.

La variable `ShadowImage` servirá para especificar la imagen asociada a la sombra a diferencia de lo que se podría pensar no se esperará recibir una plantilla de animación sino una imagen suelta. Por defecto no tendrá valor asociado para indicar que no hay imagen asociada como sombra.

A continuación se esperará leer el tipo y la clase asociada al objeto de escenario. Recordemos que en este tipo de entidades la clase está supeditada al tipo. Para leer el tipo se usará la variable `Type` y para leer la clase la variable `Class`. Estos valores serán siempre obligatorios.

Mediante la variable `IsInteractuableFlag` podremos indicar si el objeto de escenario es interactuable. Ser interactuable significará que podrá ser seleccionada por el cursor de interacción por parte del jugador y que éste podrá actuar sobre él examinándole, usando un ítem o registrándole si es un contenedor, por poner ejemplos. Por defecto un objeto de escenario será interactuable, es decir, el valor por defecto para esta variable será de `true` (no confundir con el caso de las paredes en donde el valor por defecto para esta variable es `false`).

Los últimos campos estarán destinados a establecer unos valores iniciales para los atributos locales del objeto de escenario. Estos valores se podrán a través de la variable `LocalAttribute[x]` donde x será el identificador del atributo local (definido en el archivo de reglas) sobre el que se quiere actuar. Por defecto los valores asociados a los atributos locales serán de 0.

Por defecto, todos los objetos de escenario que se coloquen sobre una celda limitarán el acceso a criaturas cuando éstas intenten desplazarse sobre la misma celda en la que éstos se encuentran. De hecho, en la etapa de diseño de áreas no será posible establecer en una misma celda tanto un objeto de escenario como una criatura. Para indicar que queremos que sobre la celda en la que se halle un objeto de escenario también pueda localizarse una criatura usaremos la variable `AccesibleFloorFlag` a la que le asociaremos `true` si deseamos tal hecho y `false` si no queremos que criatura alguna ocupe la celda del objeto de escenario. Por defecto su valor será de `false`.

Para especificar el comportamiento que deberá de tener la entidad al quedar fuera de pantalla (esto es, cuando el jugador no lo vea), se podrá escoger entre realizar una pausa o bien bajar el volumen al máximo del posible sonido asociado a su plantilla de animación. Por defecto se realizará siempre lo primero, esto es, se pausará. Al pausarse, se deberá de tener en cuenta que no se podrá variar manualmente la plantilla de animación (esto es, cambiar de estado u orientación). Si queremos indicar manualmente que queremos establecer un modo de pausa, usaremos la variable `NoVisibleInScreenMode.PauseFlag` con valor `true`, mientras que si lo que queremos es indicar que se siga con actividad, pero bajando únicamente el sonido, donde deberemos de poner `true` será en `NoVisibleInScreenMode.SilenceFlag`. En caso de poner `false` en ambas, se utilizará el modo por defecto, esto es, la pausa.

Los eventos scripts se especificarán con `ScriptEvent[x]` donde x será el identificador del script. Por defecto se entenderá que no habrá script asociado a un evento así que si no se establece nombre alguno asociado a un evento simplemente significará que no existirá script para dicho evento. A la hora de asociar un script a un evento se deberá de especificar el nombre del archivo con su extensión. Los valores de x para los eventos asociados a las paredes serán `OnObserve`, para indicar que la entidad ha sido observada, `OnTalk`, para indicar que el jugador desa hablar con la entidad, `OnManipulate`, para indicar que una entidad desea manipular a la entidad, `OnEntityCreated`, para cuando la entidad sea creada y `OnEntityIdle`, para especificar el script Idle asociado a la entidad.

Ejemplo:

```
[Profile<Columna_0>]
 Type = Columna
 Class = Piedra
 Name = Columna
 IsInteractuableFlag = true
 AnimTemplate = ATStoneColum00
End

[Profile<Baul_0>]
 Type = Baul
 Class = Normal
 Name = Baúl
 LocalAttribute[ResistenciaGolpes] = 34
 AnimTemplate = ATBaul00
 Portrait.AnimTemplate = ATBaul00
 ScriptEvent[OnManipulate] = Script_OnManipulate_Baul.cst
```

```
 AccesibleFloorFlag = true
End
```

B.6.5. El archivo de perfiles asociado a criaturas no jugador (npc's).

El archivo de perfiles de npc's (criaturas que no son jugador) es el más complejo de todos debido a la mayor cantidad datos a especificar.

El primer valor de todos estará relacionado con la plantilla de animación y será leído a través de la variable `AnimTemplate`. A diferencia de otras entidades, la plantilla de animación asociada a las criaturas (extendiendo estos requerimientos también al jugador) deberán de cumplir unas condiciones necesarias. En primer lugar, el número de orientaciones deberá de ser de 8 debido a que las criaturas podrán desplazarse y cambiando de dirección al hacerlo. El motor utilizará la orientación 0 como orientación norte, la 1 como noreste, la 2 como este, la 3 como sudeste, la 4 como sur, la 5 como sudoeste, la 6 como oeste y la 7 como noroeste. El cambio de orientación será efectuado de forma automática por el motor cuando éste tenga que mover a una criatura. El número de estados que el motor esperará que existan asociados a las plantillas de animación de una criatura será al menos de 7 ya que el motor cambiará automáticamente de estado de animación. El estado de animación 0 será usado para cuando la criatura se halle detenida, el 1 para cuando ande, el 2 para cuando corra, el 3 para cuando recoja un ítem, el 4 para cuando realice una acción de interacción, el 5 para cuando reciba un golpe y el 6 para cuando muera. Si hay problemas de poseer suficiente material gráfico, el diseñador siempre puede definir en la plantilla de animación correspondiente, todos los estados de animación iguales o bien crear sólo dos, uno para mover a la criatura (andar) y otro para mostrarla detenida. El usado para mostrarla detenida podrá ser perfectamente el utilizado también en los estados de coger, interactuar o recibir un golpe. Para los estados 0, 1, 2 y 6 el campo asociado a estado de enlace se esperará que sea el mismo ya que serán estados cíclicos. En el caso de los estados 3, 4 y 5 se esperará que el estado de enlace lleve al estado 0, para indicar que después de realizar la acción la entidad pasa a estar detenida.

A continuación se especificará el nombre la criatura. Para ello, se deberá de asignar el valor a la variable `Name`. Por defecto las criaturas no tendrán nombre asociado, pero dado que son las entidades más interactuables, será lógico que en todos los perfiles que se definan se asigne valor a la variable `Name`.

El retrato de las criaturas será definido en `Portrait.AnimTemplate`. El retrato para las criaturas será usado siempre que conversemos con ellas o bien cuando las registremos a través del interfaz de interacción con contenedores, por lo que será especialmente interesante proveer de un retrato a las criaturas. Se esperará recibir siempre un identificador de plantilla de animación.

La variable `ShadowImage` servirá para especificar la imagen asociada a la sombra. A diferencia de lo que se podría pensar no se esperará recibir una plantilla de animación sino una imagen suelta. Por defecto no tendrá valor asociado para indicar que no hay imagen asociada como sombra.

A partir de este punto se comenzará a trabajar de forma intensiva con la asociación de los identificadores definidos en el archivo de reglas relativos a las criaturas. En primer lugar se esperará recibir el tipo asociado a través de la variable `Type`. La clase de criatura se recogerá usando `Class` y el género la variable `Genre`. En ninguno de estos casos habrá valor por defecto y en todos ellos, los identificadores especificados deberán de haberse establecido en el archivo de reglas.

A continuación se esperará recibir los valores asociados al nivel, experiencia y puntos de combate asociados al perfil. En todos los casos no habrá valor por defecto y serán siempre valores numéricos. El nivel se especificará usando la variable `Level`, la experiencia la variable `Experience` y los puntos de combate la variable `CombatPoints`.

La salud y los atributos extendidos tendrán una misma forma de especificarse. Por un lado se deberá de indicar el valor base y por otro lado el valor temporal. El valor base siempre será

obligatorio mientras que el valor temporal tendrá por defecto el valor que se haya asociado al valor base. Recordemos que el valor base sirvirá para guardar el valor máximo que podrá alcanzar la criatura y el valor temporal para poder trabajar. En todos los casos se establecerán valores numéricos. Para indicar los valores de salud utilizaremos `Health.Base` y `Health.Temp` respectivamente. En el caso de los atributos extendidos usaremos las variables `ExtendedAttribute[x].Base` y `ExtendedAttribute[x].Temp` siendo `x` el valor de un identificador de atributo extendido establecido en el archivo de reglas.

Para poder establecer síntomas y habilidades en el perfil se utilizará el mismo método. Para los síntomas se utilizará el formato de variable `Symptom[x].ActiveFlag` mientras que para las habilidades `Hability[y].ActiveFlag` siendo `x` un identificador de síntoma definido en archivo de reglas e y un identificador de habilidad también definido en el archivo de reglas. Por defecto el valor será `false`, indicando que no se posee ni el síntoma ni la habilidad.

Uno de los parámetros más importantes y obligatorios, será el relativo al rango de la criatura. Por rango se entenderá el área rectangular alrededor de la criatura en la que ésta podrá detectar que otras criaturas se acercan. El valor podrá oscilar entre 0 (criatura completamente ciega y realmente sin ningún tipo de utilidad) y 10. Lo normal serán valores de 5 y 6 en el rango. La importancia del rango estará muy relacionada con el combate pues en caso de que queramos alinear a una criatura en combate sólo podremos si al menos una de las criaturas ya alineadas, independientemente de su alineación, se halla en el rango de ésta. Para establecer el rango utilizaremos la variable `Range`.

Se podrá definir que un perfil tiene el flag de transparencia activado si se asocia `true` a la variable `TransparentModeFlag`. Por defecto el valor que tendrá asociado será el de `false`. Del mismo modo, se podrá indicar que un perfil tiene activado el flag de correr usando la variable `RunModeFlag`. Por defecto, su valor será `false`.

Por defecto, todas las criaturas que se coloquen sobre una celda limitarán el acceso a otras. Para indicar que queremos que sobre la celda en la que se halle una criatura, puedan ingresar más (principalmente, de cara a trabajar con criaturas cuya elevación sea superior a cero) usaremos la variable `AccesibleFloorFlag` a la que le asociaremos `true` si deseamos tal hecho y `false` si no queremos que criatura alguna ocupe la celda. Por defecto su valor será de `false`.

Desde el archivo de perfiles se podrá indicar también si se desea que una criatura se desplace en modo fantasma. Desplazarse en modo fantasma supondrá que no tendrá en cuenta las limitaciones en los tiles y podrá atravesar paredes cerradas o violar las reglas de acceso que se hubieran podido especificar. Este flag puede resultar necesario para poder crear fantasmas o incluso para permitir que las entidades voladoras puedan simular que pasar por encima de paredes. La variable a utilizar será `GhostMoveModeFlag` y su valor por defecto será `false`.

Para especificar el comportamiento que deberá de tener la entidad al quedar fuera de pantalla (esto es, cuando el jugador no lo vea, se podrá escoger entre realizar una pausa o bien bajar el volumen al máximo del posible sonido asociado a su plantilla de animación. Por defecto se realizará siempre lo segundo, esto es, se bajará el posible sonido asociado a la animación en reproducción. Al pausarse, se deberá de tener en cuenta que no se podrá variar manualmente la plantilla de animación (esto es, cambiar de estado u orientación). Si queremos indicar manualmente que queremos establecer un modo de pausa, usaremos la variable `NoVisibleInScreenMode.PauseFlag` con valor `true`, mientras que si lo que queremos es indicar que se siga con actividad, pero bajando únicamente el sonido, donde deberemos de poner `true` será en `NoVisibleInScreenMode.SilenceFlag`. En caso de poner `false` en ambas, se utilizará el modo por defecto, esto es, la bajada de sonido al máximo.

Los eventos scripts se especificarán con `ScriptEvent[x]` donde `x` será el identificador del script. Por defecto se entenderá que no habrá script asociado a un evento así que si no se establece nombre alguno asociado a un evento simplemente significará que no existirá script para dicho evento. A la hora de asociar un script a un evento se deberá de especificar el nombre del archivo con su extensión. Los valores de `x` para los eventos asociados a las

paredes serán `OnObserve`, para indicar que la entidad ha sido observada, `OnTalk`, para indicar que el jugador desa hablar con la entidad, `OnManipulate`, para indicar que una entidad desea registrar a la entidad (sólo cuando la criatura se halle muerta, esto es, con salud inferior a la unidad), `OnEntityCreated`, para cuando la entidad sea creada, `OnEntityIdle`, para especificar el script Idle asociado a la entidad, `OnDeath`, para cuando la entidad muera, `OnResurrect`, para cuando la entidad resucite, `OnActivatedSymptom`, para cuando la entidad active un síntoma, `OnDeactivatedSymptom`, para cuando la entidad desactive un síntoma, `OnStartCombatTurn`, para cuando la entidad inicie el turno de combate, `OnUseHability`, para cuando la entidad utilice una habilidad, `OnInsertInEquipmentSlot`, para cuando la entidad equipe un ítem que llevaba en inventario sobre un slot de equipamiento, `OnRemoveFromEquipmentSlot` y para cuando la entidad elimine un ítem equipado y lo devuelva a inventario.

Ejemplo:

```
[Profile<Guardia>]
 Type = Humano
 Class = Paladin
 Genre = Varon
 Name = Owain
 Level = 12
 Experience = 21024
 CombatPoints = 9
 Health.Base = 245
 ExtendedAttribute[Destreza].Base = 12
 ExtendedAttribute[Fuerza].Base = 10
 ExtendedAttribute[Suerte].Base = 6
 Hability[Liderazgo].ActiveFlag = true
 Range = 6
 AnimTemplate = ATGuardia
 NoVisibleInScreenMode.SilenceFlag = true
 Portrait.AnimTemplate = ATCaraGuardia
 ScriptEvent[OnStartCombatTurn] = Script_OnStartCombatTurn_Guardia.cst
End
```

B.7. El archivo de especificación de áreas.

Los archivos de especificación de áreas tendrán como objetivo el diseño de un área de juego o mapa en donde deberá transcurrir la acción. Al menos deberá de existir un área de juego. Todos los archivos de áreas deberán de estar nombrados siguiendo un patrón específico ya que *CrisolBuilder* con sólo tomar un valor relativo al número de áreas que se hayan diseñado deberá de ser capaz de localizar los archivos de los mismos. Así pues, para trabajar con los archivos de áreas deberemos de tener claro que el motor las localizará siempre a partir de su identificador. El valor del identificador será siempre el número que identifique al área. Los posibles valores oscilarán siempre entre 0 y el número máximo menos uno. Sabiendo el identificador del área que vamos a editar, el nombre del archivo "`.cbt`" con el diseño del mismo será siempre "`areaID.cbt`". donde ID representará un identificador numérico. Sabiendo esto, si por ejemplo queremos diseñar un juego con 4 áreas donde podrá transcurrir la acción, los archivos que se deberán de editar tendrán por nombre "`area0.cbt`", "`area1.cbt`", "`area2.cbt`" y "`area3.cbt`". Nótese que deberán de existir todas los archivos de área. No se podrán dejar ninguno sin editar al menos en lo básico, de lo contrario *CrisolBuilder* mostrará un mensaje de error.

Antes de poder comenzar a diseñar cualquier área, los archivos de reglas y de perfiles de entidades deberán de estar muy bien construidos y diseñados. Debemos de pensar que crear un área es simplemente poner en juego nuestros diseños de personajes y reglas y que un archivo de áreas no aporta nada en sí con respecto al universo de juego; siempre se basará en todo lo que hayamos diseñado y construido con anterioridad.

B.7.1. Conceptos generales sobre la organización de un archivo de área.

Todo archivo de área tendrá tres apartados fundamentales. Por un lado el destinado a especificar la información general. Aquí se incluirán datos como la dimensión del área, su nombre o el manejo de la luz. Después pasaremos al apartado donde se especificarán los identificadores de habitación. Los identificadores de habitación servirán para que cuando asociemos dicho identificador a una o varias celdas de un área logremos que cuando el jugador pise dichas celdas los techos no se dibujen, dando la sensación de que efectivamente nos hemos metido en una habitación y andamos por dentro de la misma. El último apartado estará destinado a definir el diseño en todas y cada una de las celdas. No todas las celdas del área tendrán porque editarse. Cuando una celda no se edite el jugador jamás podrá llegar a la misma y tampoco se pintará ningún tile sobre ella. Todo lo que corresponda a su zona estará vacío (de color negro). El no editar determinadas celdas podrá ser necesario cuando estemos ante zonas nunca visibles por el jugador o para ir probando sobre la marcha que tal queda el diseño de lo que llevamos construido sin necesidad de tener que poseerlo acabado para poder visualizarlo.

Cuando el jugador entra en la habitación, los techos desaparecen

A la hora de cargar un área desde script, será necesario especificar siempre la posición que ocupará el jugador en la misma. Naturalmente, si damos una posición no válida para el área se producirá un error. Para evitar ese problema el motor, si detecta que se va a localizar al jugador en una posición que no existe (una celda no editada) buscará la primera posición libre en el área teniendo en cuenta que siempre existirá al menos una celda editada donde colocar al jugador y sin importar si en ese momento hay un objeto de escenario o una criatura. La posición alternativa donde colocar al jugador será completamente aleatoria. El diseñador verá esto en pantalla y podrá variar su diseño.

B.7.2. La sección [AreaInfo].

En esta sección se especificarán los datos de configuración general del área. En la variable `Name` deberemos de especificar el nombre del área. Resultará obligatorio especificar el nombre de un área. Las dimensiones siempre vendrán dadas en cuanto al número de celdas que tendrá el mapa. Deberemos de tener siempre en cuenta que lo normal será tener más celdas a lo alto que a lo ancho ya que *Crisol* es un motor isométrico y por cada celda a lo ancho que dibujamos, lo hacemos media vez a lo alto. En las variables `Width` y `Height` estableceremos la anchura y altura respectivamente. El valor máximo en ambos casos será de 256 y el mínimo de 1. Cuando demos valores quizás demasiado bajos, no veremos nada dibujado en el mapa ya que el motor siempre se apropiará de unos pocos tiles hacia arriba y hacia la derecha para efectuar mejores cálculos. Dichos tiles no tendrán porque tener datos ya que jamás serán vistos.

El último dato a establecer en esta sección será el relativo al modo en que será gestionada la luz en el área. Para establecer el valor luminoso ambiente se utilizará la variable `AmbientLight`. Si a esta variable le asociamos el valor 0, querremos indicar que se verá afectada por la luz ambiente de los diferentes horarios mientras que si la asociamos un valor entre 1 y 255 estaremos estableciendo un valor luminoso estático para el área, siendo el valor 1 claridad total y 255 oscuridad total. Por defecto tendrá valor 0.

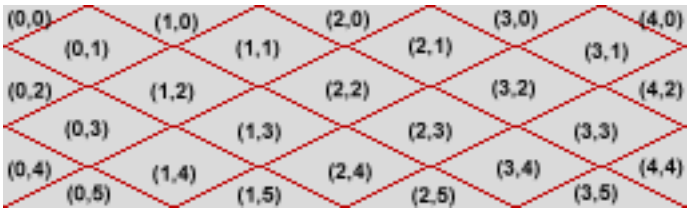
B.7.3. La sección [RoomProfiles].

El motor será capaz de gestionar habitaciones para realizar efectos relativos a la ocultación de techos para ver al jugador por debajo de los mismos. En esta sección lo que haremos será crear identificadores de perfiles de habitación que luego asociaremos a las celdas que nosotros queramos, creando conjuntos con el mismo identificador. Dichos conjuntos recibirán el nombre de habitaciones. El que una celda tenga asociada un identificador de habitación no obligará a que tenga que poseer techo. La variable que utilizaremos para ir creando los identificadores de habitación será `Room[x].Profile`, donde `x` será un valor que podrá oscilar entre `0` y el número máximo de habitaciones menos uno. Naturalmente, si no hay ninguna habitación en el área, bastará con dejar esta sección vacía.

B.7.4. Las secciones destinadas al diseño de las celdas.

El tercer apartado en la creación de un área será el destinado a diseñar las celdas que la componen. Toda sección correspondiente a este apartado tendrá siempre el mismo formato. Para empezar, el nombre de la sección se compondrá de la siguiente forma `[Cell<x,y>]` donde `x` es la coordenada en `x` de la celda e `y` la coordenada en y de la celda. Así, si queremos editar la celda 43, 21 del mapa bastará con crear la sección `[Cell<43,21>]`. Es importante tener en cuenta que no deberán de existir espacios entre los diferentes caracteres que forman el nombre e la sección.

Cuando editemos las celdas del mapa, habrá que tener presente dos cosas. Por un lado que si no editamos una celda perteneciente a una determinada coordenada, no pasará absolutamente nada pero al menos deberá de existir una celda editada para poder situar el jugador. Y por otro lado que estamos editando un motor isométrico así que aunque estemos pensando en el mapa como una matriz o un área cuadriculada, realmente deberemos de pensar en diagonal. He aquí un esquema de cómo deberán de entenderse las coordenadas:



Muestra de cómo están definidas las coordenadas en un mapa isométrico

Con esta ordenación será mucho más fácil que nuestros diseños vayan teniendo coherencia desde el principio.

Una vez que sabemos cómo definir la sección donde comenzaremos a establecer los datos de la celda que queremos diseñar, el proceso de inserción de información será siempre el mismo para todas las celdas.

En primer lugar hablaremos de cómo configurar el suelo (entidad `Floor`) asociado a la celda. Toda celda que se desee editar deberá de tener siempre un identificador de suelo asociado o de lo contrario habrá errores. El identificador de suelo se lo asignaremos siempre a la variable `Floor.Profile`. El valor asociar a esta variable será el de un identificador definido en el archivo de perfiles de suelo. A continuación se establecerá la elevación del terreno usando la variable `Floor.Elevation`. La elevación servirá para que cuando una entidad se coloque sobre el tile elevado las unicadades en pixels que se hayan establecido en la variable. Normalmente el valor a asociar será 0 (además, será lo que se asigne por defecto), pero si tenemos un tile que representa una escalera, altar o cualquier otro elemento sensible de tener altura pues deberemos de añadir más elevación. A continuación se podrán insertar ítems de tal forma que aparezcan sobre el terreno. Estos ítems que se puedan asociar aquí se les denominará ítems abandonados ya que al estar sobre el suelo carecerán de un dueño (cuando un ítem se halla en el interior de un contenedor, el dueño es ese contenedor). Para añadir ítems deberemos de usar la variable `Floor.ItemAbandoned[x].Profile` siendo x un valor que oscilará entre 0 y 255. Naturalmente, el valor a asociar aquí será el de un identificador de ítem previamente editado en el archivo de perfiles de ítems. Además de poder insertar un ítem sobre el terreno, podremos asociar a ese ítem un valor luminoso utilizando la variable definida como `Floor.ItemAbandoned[x].LightValue`. Si el valor aquí asociado es de 0, el ítem estará apagado mientras que si su valor es de 255 estará completamente encendido. Por defecto esta variable tendrá valor 0 para indicar que el ítem que se ha insertado sobre el suelo no tiene luz asociada. Por último, se podrá asociar una etiqueta al ítem de tal forma que desde *CrisolScript* se pueda obtener directamente. La etiqueta no será más que una cadena de texto no vacía que se colocará usando la variable `Floor.ItemAbandoned[x].Tag`. En caso de poner una etiqueta ya existente se producirá un error.

El siguiente paso, una vez definido el identificador de suelo asociado a la celda, será el de establecer, si se desea, el identificador de habitación al que pertenece o está asociada la misma. Para ello, tan sólo habrá que asociar un identificador de habitación, previamente creado en la sección `[RoomProfiles]`, a la variable `Room.Profile`. Por defecto no existirá ningún valor asociado a esta variable con lo que la celda no pertenecerá a habitación alguna.

Para poder especificar los techos, deberemos de tener presente que jamás se colocarán de forma automática en cuanto a altura sobre la celda, así que aunque hablemos de techos deberemos de calibrar siempre la altitud de los mismos. También se deberá de pensar que cuando estemos dibujando un techo de forma correcta y al ser el motor isométrico, no estaremos tapando al tile que tenemos debajo sino que, dependiendo de la orientación, lo haremos a los tiles a una o dos celdas de distancia o a caballo de la celda actual y la inmediatamente superior, etc. Habrá siempre que calcular de forma aproximada. También se deberá de tener en cuenta que cuando dibujamos un techo tapamos siempre a todo lo que se halle por debajo y por lo tanto desde el punto de vista de optimización del motor no nos interesará dibujar lo que se halle por debajo para hacer que *Crisol* vaya a mayor velocidad y más fluido. Este hecho se podrá indicar usando una variable lógica. Aprender a calcular qué celdas queremos que se dibujen completas aún cuando el techo se haya activo será algo que se tendrá que comprobar.

Con las explicaciones anteriores, ya podemos comentar que para asociar el identificador de techo asociado a una celda usaremos la variable `Roof.Profile`. Si no asociamos ningún identificador de techo no se buscarán las variables `Roof.Elevation`, que sirve para especificar la altura a la que se halla el techo, y `Roof.ShowUnderFlag`, que sirvirá para indicar si se desea dibujar el resto de elementos del tile cuando el techo esté activado. Su valor por defecto será `false`, lo que indicará que no se mostrará nada mientras el techo se dibuje.

Para poder establecer los objetos de escenario antes deberemos de ser conscientes de que una vez asociado a una celda, no se podrán establecer criaturas tanto durante el diseño de la celda como luego durante el juego. En caso de que más adelante se asocien datos asociados a criaturas habiéndose establecido uno o más objetos de escenario, éstos serán obviados. Dicho esto, en una celda se podrán establecer como máximo 256 objetos de escenario. Para poder establecer un objeto de escenario, siempre comenzaremos asociando a la variable `SceneObj[x].Profile` el perfil del objeto de escenario que deseamos introducir en la celda. El valor de `x` oscilará entre 0 y el número máximo de objetos de escenario que se deseen establecer menos uno (por lo que `x` podrá valer a lo sumo 255). En el instante que se lea el perfil del objeto de escenario y se de por válido, se esperará leer la elevación de éste sobre el terreno y la luz asociada. La elevación se leerá a través de la variable `SceneObj[x].Elevation` y su valor por defecto será de 0, mientras que la luz se leerá a través del valor `SceneObj[x].LightValue` y su valor por defecto será también de 0 indicando que el objeto de escenario no posee luz asociada. El valor luminoso máximo será de 255. Es importante tener en cuenta que la elevación que asociemos al objeto de escenario será independiente de la que hubiéramos asociado al suelo, es decir, que si el suelo tiene una elevación de 32 el objeto de escenario siempre se dibujará sobre el suelo con dicha altura pero que si además decimos el que el objeto de escenario tiene una elevación de, por ejemplo, 10 a esa elevación leída del suelo le añadiremos la del propio objeto de escenario. Este sistema funcionará siempre igual para el resto de entidades que puedan poseer elevación y sean insertadas en la celda. Por último, se podrá asociar una etiqueta al objeto de escenario de tal forma que desde *CrisolScript* se pueda obtener directamente. La etiqueta no será más que una cadena de texto no vacía que se colocará usando la variable `SceneObj[x].Tag`. En caso de poner una etiqueta ya existente se producirá un error.

Siempre podremos asociar ítems al objeto de escenario si éste es contenedor. En caso de que el objeto de escenario que estemos estableciendo no sea contenedor, dichos ítems serán obviados por completo durante la carga del área. El proceso para insertar ítems en un objeto de escenario contenedor será prácticamente igual que el seguido en la inserción de estos sobre el suelo de la celda. Con la variable `SceneObj[x].Content[y].Profile` podremos indicar el identificador del ítem a introducir en el contenedor, teniendo en cuenta que `y` hará referencia a un valor entre 0 y el número máximo de ítems que se podrán insertar menos uno. Dicho valor máximo será de 256 así que `y` podrá valer a lo sumo 255. A la vez que insertamos el ítem podremos añadir el posible valor luminoso asociado al mismo usando `SceneObj[x].Content[y].LightValue`. Su valor por defecto será de 0, lo que indicará que el ítem insertado no tiene luz asociada. El valor máximo posible será de 255 e indicará que el ítem insertado tiene el valor máximo de luz posible. Se dará también la oportunidad de asociar una etiqueta a los ítems que se vayan creando contenidos en el objeto de escenario. Bastará usar la variable `SceneObj[x].Content[y].Tag`.

El establecimiento de criaturas (npc's) sobre una celda estará muy limitada con respecto a la colocación de otras entidades ya que si sobre la celda en cuestión se ha establecido un objeto de escenario no se tendrá en cuenta información alguna sobre criaturas y además, sobre una celda sólo podrá existir una criatura. Con estas condiciones en mente, el primer paso para establecer una criatura será asociar el identificador de la misma. Usando la variable `Criature.Profile` podremos asociar el identificador del perfil. Una vez que se ha detectado un perfil válido se procederá a leer el resto de información, siendo el siguiente dato la elevación que la criatura tendrá. Nótese que al asignar un elevación a la criatura no estaremos hablando de la elevación que tendrá la criatura en esta celda si no la elevación que tendrá la criatura en cualquier celda. Como norma general, querremos siempre tener una elevación de 0 (a no ser que estemos creando una criatura que "levita"). Usando la variable `Criature.Elevation`

podremos establecer este valor que por defecto valdrá 0. A continuación se leerá la luz que la criatura tendrá asociada a través de la variable `Criature.LightValue`. Por defecto el valor será de 0 indicando que la criatura no tiene ningún tipo de información luminosa. El valor máximo posible será de 255 indicando luminosidad máxima. Por último, se podrá asociar una etiqueta a la criatura de tal forma que desde *CrisolScript* se pueda obtener directamente. La etiqueta no será más que una cadena de texto no vacía que se colocará usando la variable `Criature[x].Tag`. En caso de poner una etiqueta ya existente se producirá un error.

Las criaturas siempre tendrán la propiedad de ser contenedores entendiendo por contenedor el inventario que la criatura pueda transportar. Para establecer el contenido del inventario en una criatura usaremos la variable `Criature.Inventory[x].Profile` donde `x` será un valor entre 0 y 255 ya que el número máximo de ítems que una criatura podrá transportar será de 256. La variable anterior la utilizaremos para asociar el identificador de perfil de un ítem determinado. Como siempre en estos casos, todos los ítems podrán poseer un valor de luz asociado que, en este caso, se establecerá usando la variable `Criature.Inventory[x].LightValue`. Su valor por defecto será de 0 y el máximo asociable 255 el cual indicará que el ítem tiene el máximo nivel de luz posible. Desde aquí también será posible asociar una etiqueta al ítem. Bastará usar la variable `Criature.Inventory[x].Tag`.

Para concluir con la expliación relativa a la colocación de criaturas sobre el área de juego, comentar que éstas tienen la capacidad de equipar ítems en sus denominados slots de equipamiento. Los slots de equipamiento son los atributos que indican a una criatura qué posibles ítems tiene equipados. Así, toda criatura siempre tendrá, al menos, dos slots de equipamiento; el que representa la mano derecha y el que representa la mano izquierda. A la hora de definir áreas el diseñador tendrá la oportunidad, si así lo desea, de crear ítems que se asocien directamente a los slots de equipamiento de una criatura siendo el proceso, muy parecido al utilizado para establecerlos en inventario. Así, mediante las variables con prefijo `Criature.EquipmentSlot[x].`, siendo `X` el identificador de slot definido en el archivo de reglas, se podrán añadir las terminaciones `Profile`, `LightValue` y `Tag` para indicar el perfil de ítem, su posible valor de luminosidad (por defecto nulo) y su tag asociado (por defecto sin tag).

El último tipo de entidad posible de asociar sobre una celda serán las paredes. Al contrario que los objetos de escenario, criaturas y el propio suelo, en estas entidades no se podrán establecer ítems pues no actuarán de contenedores nunca. También, podrán establecerse siempre con independencia de qué otras entidades se hayan localizado en una celda. Recordemos también que el motor hará transparente siempre a aquellas paredes que estén próximas al jugador de cara a que sea visible el mismo y lo que le rodea de las paredes. También apuntar que, tal y como se comentaba en la explicación del archivo destinado a definir los perfiles de este tipo de entidades, una pared también nace con la idea de representar el concepto de puerta.

Mediante la variable `Wall[x].Profile` estableceremos el perfil de la pared que vayamos a editar. El valor de `x` podrá oscilar entre 0 y el número máximo de paredes a editar menos uno. Naturalmente, el valor asociado a esta variable será el del identificador de perfil de una pared editado en el archivo de perfiles de paredes. A continuación se esperará leer la orientación de la pared usando variables lógicas. Si la pared tiene orientación sudoeste se deberá de establecer `true` en la variable `Wall[x].SouthWestFlag`, si la tiene frontal ese valor se asociará en `Wall[x].FrontFlag` y si la tiene al sudeste `Wall[x].SouthEastFlag`. Por defecto todas estas variables tendrán valor `false` y siempre se comenzarán a examinar en el orden expuesto. En el caso de que no se halle ningún valor `true` en las variables destinadas a las orientaciones, se producirá un error. Después de la orientación, se esperará leer los valores de elevación y luz asociada a la entidad. La elevación se leerá mediante `Wall[x].Elevation` y su valor por defecto será de 0 mientras que la luz se leerá con `Wall[x].LightValue` y también su valor por defecto será 0 pudiendo alcanzar un valor máximo de 255. Las paredes también tendrán una variable más que rellenar, la última de todas, que estará destinada a indicar si el acceso a través de la misma está bloqueado o no. Esta variable se denominará `Wall[x].BlockAccessFlag` y su valor por defecto será de

`true`. Cuando una pared se halle bloqueada no será posible atravesarla desde un tile adyacente si se sigue el camino relativo a su orientación. En otras palabras, si una pared tiene orientación sudoeste, no se podrá atravesar dicha pared si se viene de una celda situado al sudoeste, si tiene orientación frontal ocurrirá lo mismo si se viene de una celda adyacente situado al sur y si se halla al sudeste ídem si se viene de una celda situada al sudeste. Por último, se podrá asociar una etiqueta a la pared de tal forma que desde *CrisolScript* se pueda obtener directamente. La etiqueta no será más que una cadena de texto no vacía que se colocará usando la variable `Wall[x].Tag`. En caso de poner una etiqueta ya existente se producirá un error.

Ejemplo:

```
[Cell<0,0>]
 Floor.Profile = Tile_2
 Room.Profile = Habitacion_1
 Roof.Profile = Techo0
 Roof.Elevation = 127
End

[Cell<4,3>]
 Floor.Profile = Tile_2
 SceneObj[0].Profile = BAUL_NORMAL_00
 SceneObj[0].Content[0].Profile = ComidaFilete_00
 SceneObj[0].Content[1].Profile = ArmaMazoPequenyo_00
 SceneObj[0].Content[2].Profile = RopaCamisa_00
 SceneObj[0].Content[3].Profile = ArmaduraCuero_00
 SceneObj[0].Content[4].Profile = ArmaduraMetal_00
 SceneObj[0].Content[5].Profile = ArmaduraPlata_00
 SceneObj[0].Content[6].Profile = CascoPlata_00
 SceneObj[0].Content[7].Profile = ArmaCuchillo_00
 SceneObj[0].Content[8].Profile = ObjetosMagicosOrbe_00
 SceneObj[0].Content[8].LightValue = 155
 SceneObj[0].Content[8].Tag = Orbe
 Room.Profile = Habitacion_1
 Roof.Profile = Techo0
 Roof.Elevation = 127
End

[Cell<10,4>]
 Floor.Profile = Tile_2
 Wall[0].SouthEastFlag = true
 Wall[0].Profile = ParedNormal_0
 Wall[0].BlockAccessFlag = true
 Room.Profile = Habitacion_1
 Roof.Profile = Techo1
 Roof.Elevation = 110
 Roof.ShowUnderFlag = true
End

[Cell<15,9>]
 Floor.Profile = TileSandWater_0
 Floor.ItemAbandoned[0].Profile = RecipienteVasija_00
End

[Cell<25,1>]
 Floor.Profile = Tile_2
 Criature[0].Profile = Guardia
 Criature[0].Tag = Guardia_1
End
```

B.8. El archivo de perfiles para la creación de un jugador.

Siempre que se vaya a comenzar una nueva partida, el jugador va a tener la oportunidad de crear su propio héroe. Las posibilidades de creación de su héroe estarán supeditadas a los parámetros que se establezcan en este archivo. En líneas generales, lo que se oferta es un conjunto de perfiles pudiendo personalizar para cada uno de ellos la forma en la que el jugador podrá repartir unos puntos con los que incrementar los valores base de los atributos que posee o bien coger habilidades. También podrá escoger el retrato que represente a su héroe según sea el género de este. Además de poder definir las posibilidades de edición del personaje, se deberán de establecer otros datos imprescindibles para que una vez que se termine de crear comience a ejecutarse el juego.

B.8.1. Sección [Profiles].

Esta será la sección en donde se definan los patrones mediante el cual se podrá crear un personaje que represente al jugador. Todos los patrones girarán en torno al valores "`Type`" que fueron identificadores definidos en el archivo de reglas. A partir de un `Type` determinado, lo que se hará será definir a su vez que valores `Class` (también definidos en el archivo de reglas) podrá escoger el jugador, cada uno de los cuales con sus propias características en cuanto a puntos de distribución y demás.

La primera serie de variables servirán para establecer los valores base que tendrá una criatura cuyo tipo sea x y su clase y en los atributos de salud, puntos de combate y atributos extendidos que se hayan definido. Será obligatorio definir siempre los valores base (esto es, los valores con los que se empieza) de salud y puntos de combate y además en estos casos, el valor que se asocie tendrá que ser superior a 0.

La variable `Type[x].Class[y].Health.BaseValue` servirá para definir el valor base de salud, la variable `Type[x].Class[y].CombatPoints.BaseValue` para hacer lo propio con el valor de puntos combate y `Type[x].Class[y].ExtendedAttribute[z].BaseValue` para establecer el valor base de un atributo extendido, con `x` un identificador `Type`, y un identificador `Class` y `z` un identificador de atributo extendido. El valor por defecto de `Type[x].Class[y].ExtendedAttribute[z].BaseValue` será 0, por lo que no será obligatorio establecer valor alguno si no se desea.

A la vez que se establecen los valores base, se podrá ir asociando el coste de poder incrementar en una unidad a los mismos. Deberemos de tener en cuenta que después existirá una variable para indicar la cantidad de puntos de distribución que para incrementar los atributos el jugador dispondrá. En caso de que asociemos un valor de 0 indicaremos que dicho atributo no podrá ser incrementado.

Mediante la variable cuyo prefijo sea `Type[x].Class[y].`, usaremos la terminación `Health.ImproveCost` para establecer el coste de incrementar el valor de salud, `CombatPoints.ImproveCost` para hacer lo propio con los puntos de combate y usando `ExtendedAttribute[z].ImproveCost` los atributos extendidos, siendo `x` un identificador `Type`, y un identificador `Class` y `z` un identificador de atributo extendido. Habrá que tener en cuenta que, aunque por defecto los valores de estas variables sean 0, en caso de asociarse nunca podrán valer negativo. Si se asocia un valor de cero, indicaremos que el atributo en cuestión no podrá aumentarse así que por defecto, ningún atributo podrá tomar puntos de distribución para ser mejorado.

A continuación se establecerá la cantidad de puntos de distribución asociados a los atributos y un flag que servirá para indicar si será obligatorio asociar todos los puntos de distribución que se oferten para poder dar por concluida la edición de un personaje. La variable `Type[x].Class[y].DistributionPoints.Attributes.Value` establecerá la cantidad de puntos de distribución que hay disponibles, su valor por defecto será de 0, y la variable `Type[x].Class[y].DistributionPoints.Attributes.MustUseFlag` indicará si será necesario asociar todos los puntos de distribución ofertados. Su valor por defecto será `false`.

Una vez configurado el trabajo con los atributos, se deberá de hacer lo propio con las habilidades. En este caso, habrá que tener en cuenta que asociar una habilidad deberá de tener siempre un coste igual o superior a 0 y que para ello, al igual que con los atributos, dispondremos de una serie de puntos para invertir y un valor lógico que podremos establecer a `true` o `false` para indicar si es absolutamente necesario gastar todos esos puntos que se nos ofertan. Cuando asociemos un coste de 0 para una habilidad, realmente estaremos indicando que dicha habilidad no podrá ser establecida bajo ningún concepto al perfil de criatura que se está creando. Mediante la variable `Type[x].Class[y].Hability[z].SetCost` estableceremos el coste, en puntos, que el jugador deberá de hacer para conseguir tener la habilidad `z`. El valor de `x` representa un identificador `Type`, el `y` uno `Class` y el `z` uno `Hability`. En todos los casos, dichos identificadores han tenido que haber sido definidos en el archivo de reglas para las criaturas.

A continuación se establecerán la cantidad de puntos de distribución asociados a las habilidades y un flag que servirá para indicar si será obligatorio asociar todos los puntos de distribución que se oferten para poder dar por concluida la edición de un personaje. La variable `Type[x].Class[y].DistributionPoints.Habilities.Value` establecerá la cantidad de puntos de distribución que hay disponibles, su valor por defecto será de 0, y la variable `Type[x].Class[y].DistributionPoints.Habilities.MustUseFlag` indicará si será necesario asociar todos los puntos de distribución ofertados. Su valor por defecto será `false`.

Ejemplo:

```
[Profiles]
 Type[Humano].Class[Paladin].Health.BaseValue = 55
 Type[Humano].Class[Paladin].Health.ImproveCost = 1
 Type[Humano].Class[Paladin].CombatPoints.BaseValue = 6
 Type[Humano].Class[Paladin].CombatPoints.ImproveCost = 0
 Type[Humano].Class[Paladin].ExtendedAttribute[Destreza].BaseValue = 8
 Type[Humano].Class[Paladin].ExtendedAttribute[Destreza].ImproveCost = 2
 Type[Humano].Class[Paladin].ExtendedAttribute[Fuerza].BaseValue = 8
 Type[Humano].Class[Paladin].ExtendedAttribute[Fuerza].ImproveCost = 1
 Type[Humano].Class[Paladin].ExtendedAttribute[Suerte].BaseValue = 4
 Type[Humano].Class[Paladin].ExtendedAttribute[Suerte].ImproveCost = 2

 Type[Humano].Class[Paladin].Hability[Magia].SetCost = 1
 Type[Humano].Class[Paladin].DistributionPoints.Attributes.Value = 10
 Type[Humano].Class[Paladin].DistributionPoints.Attributes.MustUseFlag = false
 Type[Humano].Class[Paladin].DistributionPoints.Habilities.Value = 1
 Type[Humano].Class[Paladin].DistributionPoints.Habilities.MustUseFlag = false
 Type[Humano].Class[Mago].Health.BaseValue = 30
 Type[Humano].Class[Mago].Health.ImproveCost = 3
 Type[Humano].Class[Mago].CombatPoints.BaseValue = 5
 Type[Humano].Class[Mago].CombatPoints.ImproveCost = 0
 Type[Humano].Class[Mago].ExtendedAttribute[Destreza].BaseValue = 7
 Type[Humano].Class[Mago].ExtendedAttribute[Destreza].ImproveCost = 1
 Type[Humano].Class[Mago].ExtendedAttribute[Fuerza].BaseValue = 6
 Type[Humano].Class[Mago].ExtendedAttribute[Fuerza].ImproveCost = 4
 Type[Humano].Class[Mago].ExtendedAttribute[Suerte].BaseValue = 6
 Type[Humano].Class[Mago].ExtendedAttribute[Suerte].ImproveCost = 1
 Type[Humano].Class[Mago].Hability[Magia].SetCost = 1

 Type[Humano].Class[Mago].DistributionPoints.Attributes.Value = 5
 Type[Humano].Class[Mago].DistributionPoints.Attributes.MustUseFlag = false
 Type[Humano].Class[Mago].DistributionPoints.Habilities.Value = 19
 Type[Humano].Class[Mago].DistributionPoints.Habilities.MustUseFlag = false

 Type[Humano].Sprite.Genre[Varon].AnimTemplate = ATCriaturePlayer0A
 Type[Humano].Sprite.Genre[Varon].ShadowImage = Graphics\CriaturePlayer0Ashadow.tga
 Type[Humano].Sprite.Genre[Hembra].AnimTemplate = ATCriaturePlayer0B
 Type[Humano].Sprite.Genre[Hembra].ShadowImage = Graphics\CriaturePlayer0Bshadow.tga

 Type[Hobbit].Class[Explorador].Health.BaseValue = 25
 Type[Hobbit].Class[Explorador].Health.ImproveCost = 1
 Type[Hobbit].Class[Explorador].CombatPoints.BaseValue = 1
 Type[Hobbit].Class[Explorador].CombatPoints.ImproveCost = 2
 Type[Hobbit].Class[Explorador].ExtendedAttribute[Destreza].BaseValue = 6
 Type[Hobbit].Class[Explorador].ExtendedAttribute[Destreza].ImproveCost = 2
 Type[Hobbit].Class[Explorador].ExtendedAttribute[Fuerza].BaseValue = 4
 Type[Hobbit].Class[Explorador].ExtendedAttribute[Fuerza].ImproveCost = 2
 Type[Hobbit].Class[Explorador].ExtendedAttribute[Suerte].BaseValue = 8
```

```
 Type[Hobbit].Class[Explorador].ExtendedAttribute[Suerte].ImproveCost = 2
 Type[Hobbit].Class[Explorador].Hability[Magia].SetCost = 0

 Type[Hobbit].Class[Explorador].DistributionPoints.Attributes.Value = 13
 Type[Hobbit].Class[Explorador].DistributionPoints.Attributes.MustUseFlag = false
 Type[Hobbit].Class[Explorador].DistributionPoints.Habilities.Value = 1
 Type[Hobbit].Class[Explorador].DistributionPoints.Habilities.MustUseFlag = false

End
```

B.8.2. Sección [Portrait].

En esta sección se establecerán los conjuntos de retratos posibles según género. En pantalla, al cambiar el género de la criatura que se esté creando el conjunto de retratos cambiará también de forma automática. Al menos deberá de existir un retrato para cada género. La variable a utilizar será `Genre[x].Portrait[y].AnimTemplate` donde `x` será el identificador de uno de los dos posibles géneros existentes e y un valor numérico que oscilará entre 0 y 255. Naturalmente, se esperará que se asigne el nombre del identificador de una plantilla de animación previamente definido en el archivo de plantillas de animación.

Asociados a los retratos, se establecerán los sprites en que representarán al jugador, esto es, la representación del personaje que representará al usuario en pantalla. La variable a utilizar será Genre`[x].Portrait[y].Player.AnimTemplate` siendo `x` el identificador del género e y el identificador numérico del retrato. También se podrá definir el gráfico con la sombra asociada si bien, no será obligatorio. Tal valor se indicará con la variable `Genre[x].Portrait[y].Player.ShadowImage`. Necesariamente se deberá de poner el identificador del género 0 y 1 (ya que sólo habrá dos géneros posibles). En el caso de la sombra, no existirá por defecto así que si no se indica, ese sprite no tendrá asociada. Nótese que por cada retrato que se establezca se deberá de indicar una representación para el jugador.

Ejemplo:

```
[Portraits]
 Genre[Varon].Portrait[0].AnimTemplate = ATCaraVaron0
 Genre[Varon].Portrait[0].Player.AnimTemplate = ATVaron0
 Genre[Varon].Portrait[1].AnimTemplate = ATCaraVaron1
 Genre[Varon].Portrait[1].Player.AnimTemplate = ATVaron1
 Genre[Varon].Portrait[2].AnimTemplate = ATCaraVaron2
 Genre[Varon].Portrait[2].Player.AnimTemplate = ATVaron2
 Genre[Hembra].Portrait[0].AnimTemplate = ATCaraHembra0
 Genre[Hembra].Portrait[0].AnimTemplate = ATHembra0
 Genre[Hembra].Portrait[1].AnimTemplate = ATCaraHembra1
 Genre[Hembra].Portrait[1].AnimTemplate = ATHembra1
End
```

B.8.3. Sección [Common].

En esta sección se establecerán los datos comunes, independientemente de la criatura que se haya editado en cuanto a tipo y clase o retrato escogido.

La primera variable hará referencia al identificador del área que se utilizará como área de inicio, esto es, el lugar donde comenzará el juego el jugador. La variable será `StartLocation.IDArea` y esperará recibir un valor asociado a un identificador de área existente.

A continuación se esperará recibir las coordenadas en donde se situará al jugador en dicho área. Naturalmente, esas coordenadas deberán de corresponder a un tile válido dentro del área. Para establecerlas se utilizarán las variables `StartLocation.Cell.XPos` y `StartLocation.Cell.YPos`.

Los eventos scripts se especificarán con `ScriptEvent[x]` donde `x` será el identificador del script. Por defecto se entenderá que no habrá script asociado a un evento así que si no se establece nombre alguno asociado a un evento simplemente significará que no existirá script

para dicho evento. A la hora de asociar un script a un evento se deberá de especificar el nombre del archivo con su extensión. Los valores de x para los eventos asociados a las paredes serán `OnObserve`, para indicar que la entidad ha sido observada, `OnTalk`, para indicar que el jugador desa hablar con la entidad, `OnManipulate`, para indicar que una entidad desea manipular a la entidad, `OnEntityCreated`, para cuando la entidad sea creada, `OnEntityIdle`, para especificar el script Idle asociado a la entidad, `OnGet`, para cuando la entidad sea recogida, `OnDrop`, para cuando la entidad sea soltada, `OnUseItem`, para cuando la entidad sea utilizada, `OnTradeItem`, para cuando la entidad sea intercambiada, `OnHit`, para cuando la entidad haya sido golpeada, `OnDeath`, para cuando la entidad muera, `OnResurrect`, para cuando la entidad resucite, `OnActivatedSymptom`, para cuando la entidad active un síntoma, `OnDeactivatedSymptom`, para cuando la entidad desactive un síntoma, `OnStartCombatTurn`, para cuando la entidad inicie el turno de combate, `OnEndCombatTurn`, para cuando la entidad finalice su turno de combate, `OnUseHability`, para cuando la entidad utilice una habilidad, `OnInsertInEquipmentSlot`, para cuando la entidad equipe un ítem que llevaba en inventario sobre un slot de equipamiento, `OnRemoveFromEquipmentSlot` y para cuando la entidad elimine un ítem equipado y lo devuelva a inventario.

Ejemplo:

```
[Common]
 StartLocation.IDArea = 0
 StartLocation.Cell.XPos = 15
 StartLocation.Cell.YPos = 34
 ScriptEvent[OnObserve] = Script_OnObserve_Player.cst
 ScriptEvent[OnUseHability] = Script_OnUseHability_Player.cst
End
```

B.9. El archivo de configuración de interfaces.

Este archivo de datos será el más importante a la hora de personalizar el motor desde el punto de vista visual pues en él se definirá la configuración de los interfaces que se mostrarán en pantalla así como los cursores y selectores. Por interfaz entenderemos cualquier pantalla o menú informativo del motor, cuadros de diálogo, ventanas de inventario del jugador, para la gestión de partidas guardadas, etc.

B.9.1. Conceptos generales.

La construcción de los interfaces se basará de forma intensiva en la colocación de botones, imágenes de fondo, posición de texto, color, etc. Siempre que se hable de botones se pedirá una plantilla de animación que deberá de tener dos estados; uno representando al botón sin seleccionar (estado de animación 0) y otro representando al botón seleccionado (estado de animación 1). Naturalmente, la plantilla de animación podrá tener más estados pero el motor siempre irá a buscar el estado 0 para indicar que el botón no se halla seleccionado y el 1 para indicar que sí lo está. Si nosotros a partir de esos estados queremos enlazar con otro podremos, pero sabiendo de la existencia de los mencionados y de cómo serán utilizados. Todos los botones serán configurados, a no ser que se indique lo contrario, utilizando tres variables. La primera de ellas para para asociar la plantilla de animación y las dos últimas para establecer la posición en donde será dibujado el botón. La variable en la que se asociará la plantilla de animación acabará siempre en `AnimTemplate` y las relativas a las coordenada en x e y en `XPos` e `YPos`.

En algunos casos se hablará de un elemento gráfico denominado `FXBack`. Dicho elemento podrá estar asociado a texto o a selectores y no será más que un rectángulo de un determinado color RGB y grado de transparencia. En ambos casos estos valores serán completamente configurables. Como ejemplo de qué es un `FXBack`, se muestra la siguiente figura:

Parte de un interfaz de juego principal con la muestra de tres componentes FXBack.

En la imagen anterior, se pueden ver, señalados, tres componentes `FXBack`. El primer de ellos es el que se halla por debajo del texto de la consola de juego, de un color marrón claro, mientras que los otros dos pertenecen a un selector principal (mano) y al retrato del jugador respectivamente (en ambos casos, el color de fondo es azulado).

Habrá que tener en cuenta también, que dependiendo de si el juego se diseña en una resolución u otra, el motor no hará el trabajo de situar de forma automática los interfaces, es por ello que si diseñamos el juego en una resolución será para jugarlo en esa resolución y por tanto, todo el diseño de interfaces en cuanto a posición deberá de estar pensando para dicha resolución.

B.9.2. Sección [Cursors].

En esta sección se establecerán las plantillas de animación asociadas a los cursores del motor. Cuando hablamos de los cursores del motor nos referimos a los iconos que representarán al ratón. Dependiendo del tipo de cursor de que se esté hablando las exijencias en cuanto a la plantilla de animación que se asocie podrán variar.

El primer cursor de todos será el relativo al movimiento, esto es, el cursor que señale tiles y que indique de alguna forma visual si es posible moverse o no a una determinada posición. Dicho

cursor se asociará usando la variable `Movement.AnimTemplate`. El cursor deberá de tener al menos 2 estados. El estado 0 se establecerá siempre que sea posible moverse a dicho tile mientras que el estado 1 se convertirá en activo cuando no lo sea.

El cursor de interacción con entidades se establecerá a través de la variable `EntityInteract.AnimTemplate`. La plantilla que se asocie deberá de tener al menos dos estados. El estado 0 se utilizará para indicar que es posible interactuar con la entidad en cuestión mientras que el estado 1 servirá para indicar que no lo es.

Cuando lo que se quiera sea interactuar con una entidad pero para golpearla el cursor también podrá ser distinto para ello, la variable `EntityHit.AnimTemplate` recogerá el identificador de la plantilla de animación que represente al cursor de interacción para golpear. Dicha animación deberá de tener al menos dos estados. El estado 0 servirá para indicar que es posible golpear a la entidad mientras que el estado 1 será asociado para indicar justamente lo contrario.

En los momentos de espera (cuando durante el combate se esté esperando a que los enemigos controlados por el motor realicen los movimiento) se podrá asociar un cursor distinto. Dicho cursor se indicará a través de la variable `Waiting.AnimTemplate` que esperará recibir una plantilla de animación con al menos un estado. El motor siempre asociará el estado de animación 0.

Cuando se realice scroll sobre el decorado (al poner el cursor en los extremos de la pantalla) el cursor también podrá ser distinto con el fin de indicar que se está realizando scroll. En este caso, el identificador de plantilla de animación se recogerá a través de la variable `MapScroll.AnimTemplate`. La animación además deberá de poseer un total de 8 estados siendo el estado 0 el utilizado cuando se realice scroll al norte, el 1 cuando sea al noreste, el 2 cuando sea al este, el 3 cuando sea al sudeste, el 4 cuando se haga al sur, el 5 cuando se haga al sudoeste, el 6 al oeste y el 7 al noroeste.

Ejemplo de un set de cursores de scroll

El último cursor de todos estará destinado a la interacción con las ventanas de interfaz, esto es, menús, cuadros de diálogo, etc. Se recogerá a través de la variable `GUIInteract.AnimTemplate` y la plantilla de animación debera de poseer al menos un estado de animación pues el motor siempre establecerá el estado 0 al activarse este cursor.

Ejemplo:

```
[Cursors]
 Movement.AnimTemplate = ATGUIMovementCursor
 EntityInteract.AnimTemplate = ATGUIEntityInteractCursor
 EntityHit.AnimTemplate = ATGUIEntityInteractForHitCursor
 Waiting.AnimTemplate = ATGUIWaitingCursor
 MapScroll.AnimTemplate = ATGUIIsoScrollCursor
 GUIInteract.AnimTemplate = ATGUIWindowInteractCursor
End
```

B.9.3. Sección [Selectors].

Esta sección estará dedicada a establecer los selectores que serán utilizados en el motor. Cuando hablemos de selectores haremos referencia a las animaciones que se dibujarán sobre una celda y debajo de todas las entidades que se hallen por encima. Habrá dos casos en lo que esto ocurra, mientras se esté ordenando movimientos y durante el combate asociando selectores distintos dependiendo del bando en el que las criaturas que se hallen en la lista de combate estén alineadas.

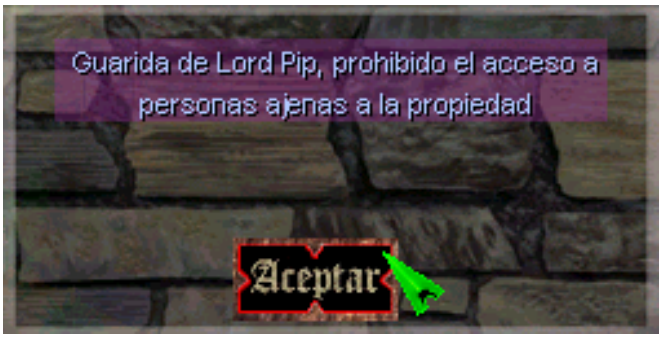
Así, para indicar el selector a utilizar cuando se estén realizando movimientos, se utilizará la variable `Map.AnimTemplate`. El identificador de la plantilla de animación que se espera

deberá de poseer al menos un estado de animación. El motor establecerá siempre el estado 0 cuando active dicho selector.

Para los selectores de combate tendremos dos variables, por un lado `Combat.PlayerAlingment.AnimTemplate` para asociar el identificador de plantilla de animación asociada a las criaturas que se hallen del lado del jugador durante el combate y `Combat.EnemyPlayerAlingment.AnimTemplate` para hacer lo propio con las criaturas que se hallen del lado de los enemigos del jugador durante el combate. En ambos casos, se esperará que las plantillas de animación posean al menos un estado ya que el motor establecerá el estado 0 al activarlos.

B.9.4. Sección [AdviceDialog].

En esta sección se establecerá la configuración del cuadro de diálogo que se mostrará cuando se quiera enseñar un aviso o una notificación de algún tipo.



Ejemplo de un cuadro de diálogo de aviso

Se utilizará `BackImage.FileName` para obtener el gráfico que actuará de imagen de fondo, `BackImage.XPos` y `BackImage.YPos` para leer la posición en pantalla donde comenzará a dibujarse dicha imagen y `BackImage.Alpha` para establecer el valor alpha o de transparencia a la imagen de fondo. Nótese que el cuadro de diálogo se superpondrá siempre a cualquier elemento que esté por debajo y contar con un cierto grado de transparencia siempre dará un aspecto mucho más interesante a la imagen en pantalla. El valor 0 será transparencia total y el de 255 solidez total.

A continuación se esperará leer el formato con el que el texto que contenga la descripción del aviso o notificación será dibujado. Su posición se leerá a través de las variables `TextFormat.XPos` y `TextFormat.YPos` mientras que el área a lo ancho destinada a escribir el texto se hará usando la variable `TextFormat.AreaWidth`. En caso de que el texto no quepa en ese área se formateará de tal manera que ocupe las líneas que hagan falta. El valor mínimo para el área será de 64 y el máximo de 256. El color RGB del texto se establecerá con las variables con prefijo `TextFormat.` y ternubacuibes `RedColor`, `BlueColor` y `GreenColor` respectivamente. Por último, antes del texto se dibujará una especie de efecto especial denominado FXBack destinado a resaltarlo mejor. Dicho efecto especial tendrá un color RGB asociado que se establecerá usando las variables con prefijo `TextFormat.FXBack.` y terminaciones `RedColor`, `GreenColor` y `BlueColor` respectivamente, además de un valor de transparencia para que no aparezca nunca de forma brusca a través de las variable `TextFormat.FXBack.Alpha`.

El último apartado estará destinado a establecer el botón que hará de opción de aceptación. Dicho de otra forma, al pulsar el botón de aceptación se entenderá que el aviso notificado por el cuadro de diálogo ha sido leido y se desea cerrar el mismo. Para establecer la plantilla de animación se usará la varible `Options.AcceptButton.AnimTemplate` mientras que la posición en donde aparecerá dicha opción o botón será localizada usando las variables `Options.AcceptButton.XPos` y `Options.AcceptButton.YPos`.

Ejemplo:

```
[AdviceDialog]
```

```
 BackImage.FileName = Graphics\Interfaz\BackImgDialog.tga
 BackImage.XPos = 195
 BackImage.YPos = 130
 BackImage.Alpha = 225
 TextFormat.XPos = 220
 TextFormat.YPos = 145
 TextFormat.AreaWidth = 210
 TextFormat.RedColor = 180
 TextFormat.GreenColor = 200
 TextFormat.BlueColor = 255
 TextFormat.FXBack.RedColor = 150
 TextFormat.FXBack.GreenColor = 50
 TextFormat.FXBack.BlueColor = 150
 TextFormat.FXBack.Alpha = 100
 Options.AcceptButton.AnimTemplate = ATAcceptButton
 Options.AcceptButton.XPos = 285
 Options.AcceptButton.YPos = 220
End
```

B.9.5. Sección [QuestionDialog].

Esta sección servirá para especificar la configuración del cuadro de diálogo destinado a recabar información del usuario a través de una simple pregunta. Se diferenciará del cuadro de diálogo de aviso en que éste incorporará dos botones de opciones (uno para aceptar lo que se pueda preguntar y otro para rechazarlo) más luego un botón opcional para cancelar la pregunta.

Ejemplo de un cuadro de diálogo de pregunta

En primer lugar se esperará leer la configuración de la imagen de fondo a través de `BackImage.FileName` para obtener el gráfico que actuará de imagen de fondo, `BackImage.XPos` y `BackImage.YPos` para leer la posición en pantalla donde comenzará a dibujarse dicha imagen y `BackImage.Alpha` para establecer el valor alpha o de transparencia de la imagen. El valor 0 será transparencia total y el de 255 solidez total.

A continuación se esperará leer el formato con el que el texto que contenga la pregunta será dibujado. Su posición se leerá a través de las variables `TextFormat.XPos` y `TextFormat.YPos` mientras que el área a lo ancho destinada a escribir el texto se hará usando la variable `TextFormat.AreaWidth`. En caso de que el texto no quepa en esa área se formateará de tal manera que ocupe las líneas que hagan falta. El valor mínimo para el área será de 64 y el máximo de 256. El color RGB del texto se establecerá con las variables con prefijo `TextFormat.`, y terminaciones `RedColor`, `BlueColor` y `GreenColor` respectivamente. Por último, antes del texto se dibujará una especie de efecto especial denominado FXBack destinado a resaltarlo mejor. Dicho efecto especial tendrá un color RGB asociado que se establecerá usando las variables con prefijo `TextFormat.FXBack.` y terminaciones `RedColor`, `GreenColor` y `BlueColor` respectivamente, además de un valor de transparencia para que no aparezca nunca de forma brusca a través de las variable `TextFormat.FXBack.Alpha`.

Acto seguido se deberá de leer la configuración asociada a los botones de aceptación y rechazo. La configuración del botón aceptación de la pregunta se leerá a través de la variable

`Options.YesButton.AnimTemplate` para leer la plantilla de animación y `Options.YesButton.XPos` y `Options.YesButton.YPos` la posicón en la que será dibujada. Por otro lado, la variable `Options.NoButton.AnimTemplate` servirá para leer el identificador de la plantilla de animación asociada al botón de rechazo de la pregunta. Completando esta información estarán las variables `Options.NoButton.XPos` y `Options.NoButton.YPos` para indicar la posición en la que serán dibujadas.

La última información será relativa al botón de cancelación. En la variable `Options.CancelButton.AnimTemplate` se recogerá el perfil del botón de cancelación mientras que en las variables con prefijo `Options.CancelButton.` y terminación de `XPos` y `YPos` respectivamente, las coordenadas de dibujado de dicho botón.

Ejemplo:

```
[QuestionDialog]
 BackImage.FileName = Graphics\Interfaz\BackImgDialog.tga
 BackImage.XPos = 195
 BackImage.YPos = 130
 BackImage.Alpha = 225

 TextFormat.XPos = 220
 TextFormat.YPos = 145
 TextFormat.AreaWidth = 210
 TextFormat.RedColor = 180
 TextFormat.GreenColor = 200
 TextFormat.BlueColor = 255
 TextFormat.FXBack.RedColor = 150
 TextFormat.FXBack.GreenColor = 50
 TextFormat.FXBack.BlueColor = 150
 TextFormat.FXBack.Alpha = 100

 Options.YesButton.AnimTemplate = ATYesButton
 Options.YesButton.XPos = 235
 Options.YesButton.YPos = 220
 Options.NoButton.AnimTemplate = ATNoButton
 Options.NoButton.XPos = 345
 Options.NoButton.YPos = 220
 Options.CancelButton.AnimTemplate = ATCancelButton
 Options.CancelButton.XPos = 275
 Options.CancelButton.YPos = 220
End
```

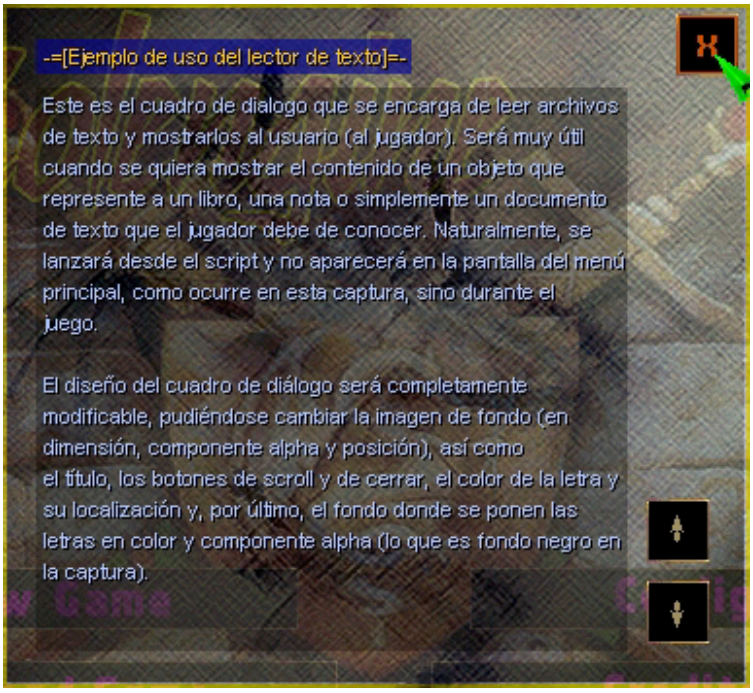
B.9.6. Sección [TextFileReader].

En esta sección se definirá la configuración del interfaz destinado a mostrar archivos de texto. Este interfaz estará provisto de dos botones de scroll para moverse por el fichero leído, otro destinado al cierre de interfaz, una imagen de fondo asociada al interfaz, todo lo relativo a la configuración para indicar indicar como será mostrando el texto y el título asociado a la interfaz.

En primer lugar se esperará leer la configuración de la imagen de fondo. A través de `BackImage.FileName` se leerá el gráfico que actuará de imagen de fondo, `BackImage.XPos` y `BackImage.YPos` servirá para leer la posición en pantalla donde comenzará a dibujarse dicha imagen y `BackImage.Alpha` para establecer el valor alpha o de transparencia de la misma. El valor 0 será transparencia total y el de 255 solidez total.

El siguiente paso será el de establecer el formato con el que se mostrará el título del interfaz. Su posición se leerá a través de las variables `TitleFormat.XPos` y `TitleFormat.YPos` mientras que el área a lo ancho destinada a escribir el título se hará usando la variable `TitleFormat.AreaWidth`. En caso de que el texto no quepa en esa área se formateará de tal manera que ocupe las líneas que hagan falta. El valor mínimo para el área será de 64 y el máximo de 256. El color RGB del texto se establecerá con las variables con prefijo `TextFormat.`, de las terminaciones `RedColor`, `BlueColor` y `GreenColor` respectivamente. Por último, antes del texto se dibujará una especie de efecto especial denominado FXBack

destinado a resaltarlo mejor. Dicho efecto especial tendrá un color RGB asociado que se establecerá usando las variables con prefijo `TextFormat.FXBack.` y terminación `RedColor`, `GreenColor` y `BlueColor` respectivamente, además de un valor de transparencia para que no aparezca nunca de forma brusca a través de las variable `TextFormat.FXBack.Alpha`.



Ejemplo de un cuadro de diálogo destinado a la lectura de un archivo de texto

A continuación se esperará leer toda la información relativa al formato de dibujado del fichero de texto que se lea. En líneas generales, los datos a establecer serán los mismos que para el título solo que usando variables distintas. Las coordenadas de posición donde comenzará a mostrarse la primera línea de texto se leerán de las variables `TextFileFormat.XPos` y `TextFileFormat.YPos` mientras que el área ocupado por el texto en pixels se leerá a través de la variable `TextFileFormat.AreaWidth`. Para indicar el número máximo de líneas que podrán mostrarse se utilizará la variable `TextFileFormat.MaxLines` siendo su valor mínimo de 1 y el valor máximo 256. El color RGB del texto asociado al fichero de texto leido se establecerá a través de las variables con prefijo `TextFileFormat.` y seguidas de `RedColor`, `GreenColor` y `BlueColor` respectivamente, mientras que la configuración del FXBack relativa a su color RGB y su nivel alpha comenzarán todas con el prefijo `TexFileFormat.FXBack.` seguido de `RedColor`, `GreenColor`, `BlueColor` y `Alpha` respectivamente.

Ejemplo:

```
[TextFileReader]
 BackImage.FileName = Graphics\Interfaz\TextReaderBackImg.tga
 BackImage.XPos = 114
 BackImage.YPos = 20
 BackImage.Alpha = 175

 TitleFormat.XPos = 135
 TitleFormat.YPos = 40
 TitleFormat.RedColor = 255
 TitleFormat.GreenColor = 200
 TitleFormat.BlueColor = 0
 TitleFormat.FXBack.RedColor = 0
 TitleFormat.FXBack.GreenColor = 0
 TitleFormat.FXBack.BlueColor = 155
 TitleFormat.FXBack.Alpha = 155

 TextFileFormat.XPos = 135
 TextFileFormat.YPos = 65
```

```
 TextFileFormat.AreaWidth = 300
 TextFileFormat.MaxLines = 18
 TextFileFormat.RedColor = 180
 TextFileFormat.GreenColor = 200
 TextFileFormat.BlueColor = 255
 TextFileFormat.FXBack.RedColor = 0
 TextFileFormat.FXBack.GreenColor = 0
 TextFileFormat.FXBack.BlueColor = 0
 TextFileFormat.FXBack.Alpha = 75

 Options.UpButton.AnimTemplate = ATUpButton
 Options.UpButton.XPos = 445
 Options.UpButton.YPos = 275
 Options.DownButton.AnimTemplate = ATDownButton
 Options.DownButton.XPos = 445
 Options.DownButton.YPos = 317
 Options.ExitButton.AnimTemplate = ATXButton
 Options.ExitButton.XPos = 460
 Options.ExitButton.YPos = 25
End
```

B.9.7. Sección [TextOptionsSelector].

Esta sección servirá para configurar el interfaz destinado a mostrar opciones de texto con el fin de elegir uno de ellas.

Ejemplo de un cuadro de selección de texto

En primer lugar se esperará leer la configuración de la imagen de fondo. Con la variable `BackImage.FileName` se indicará el nombre de la imagen de fondo y mediante, `BackImage.XPos` y `BackImage.YPos` la posición en donde comenzará a ser dibujada. Por último, con `BackImage.Alpha` se establecerá el grado de transparencia, pudiendo asociar como máximo un valor de 255 que es solidez absoluta.

El siguiente paso será el de establecer el formato con el que se mostrará el título del interfaz. Todas las variables comenzarán por `TitleFormat.`. Así, su posición se leerá añadiendo la terminación `XPos` e `YPos` mientras que el área a lo ancho destinada a escribir el título se hará usando la terminación `AreaWidth`. El valor mínimo para el área será de 64 y el máximo de 256. El color RGB del texto se establecerá con las terminaciones `RedColor`, `GreenColor` y `BlueColor`. Por último, antes del texto se dibujará el FXBack, que deberá de configurarse en color y valor de transparencia a través de las variables cuya terminación sea `FXBack.RedColor`, `FXBack.GreenColor`, `FXBack.BlueColor` y `FXBack.Alpha` respectivamente.

A continuación vendrá la configuración de las líneas de opciones de texto. Habrá que tener en cuenta que por un lado deberemos de configurar las líneas para cuando estén y no estén seleccionadas. Para ello, únicamente habrá que indicar el color de las mismas pues la idea es que el color cambie dependiendo de si son o no señaladas con el cursor. La posición donde comenzará a dibujarse la primera línea de texto, así como el número máximo de ellas que se mostrarán simultáneamente y el área que ocuparán en pixels serán establecidas a través de

las variables `TextOptionsFormat.XPos`, `TextOptionsFormat.YPos` para la posición, `TextOptionsFormat.MaxLines` para indicar el número de líneas y `TextOptionsFormat.AreaWidth` para la anchura del área. Para especificar el color RGB de las líneas cuando éstas sean seleccionadas, se utilizarán las variables que comiencen por el prefijo `TextOptionsFormat.Select.` y sigan con `RedColor`, `GreenColor` y `BlueColor` respectivamente, mientras que para especificar el color que tendrán cuando no sean seleccionadas por el cursor (sólo podrá existir una línea seleccionada por el cursor de forma simultánea) se usarán las variables de prefijo `TextOptionsFormat.Unselect.` seguidas de `RedColor`, `GreenColor` y `BlueColor` respectivamente.

El siguiente y último paso será configurar los botones de opciones. Dos de ellos estarán destinados a mover líneas arriba y abajo y el tercero a cerrar el interfaz cancelando la acción de seleccionar una opción de texto. Como siempre, la configuración de un botón se realizará especificando el identificador de la plantilla de animación y su posición de dibujado. Así, la configuración del botón de desplazamiento de líneas hacia arriba se hará usando las variables con prefijo `Options.UpButton.` y las de desplazamiento hacia abajo `Options.DownButton.` seguidas, en ambos casos, de `XPos`, `YPos` y `AnimTemplate` para indicar la posición y la plantilla de animación asociada respectivamente.

Ejemplo:

```
[TextOptionsSelector]
 BackImage.FileName = Graphics\Interfaz\OptionsSelectorBackImg.tga
 BackImage.XPos = 124
 BackImage.YPos = 70
 BackImage.Alpha = 225

 TitleFormat.XPos = 195
 TitleFormat.YPos = 85
 TitleFormat.RedColor = 255
 TitleFormat.GreenColor = 200
 TitleFormat.BlueColor = 0
 TitleFormat.FXBack.RedColor = 0
 TitleFormat.FXBack.GreenColor = 0
 TitleFormat.FXBack.BlueColor = 155
 TitleFormat.FXBack.Alpha = 155

 TextOptionsFormat.XPos = 145
 TextOptionsFormat.YPos = 120
 TextOptionsFormat.MaxLines = 6
 TextOptionsFormat.AreaWidth = 300
 TextOptionsFormat.Select.RedColor = 255
 TextOptionsFormat.Select.GreenColor = 255
 TextOptionsFormat.Select.BlueColor = 255
 TextOptionsFormat.Unselect.RedColor = 180
 TextOptionsFormat.Unselect.GreenColor = 180
 TextOptionsFormat.Unselect.BlueColor = 200
 TextOptionsFormat.FXBack.RedColor = 0
 TextOptionsFormat.FXBack.GreenColor = 0
 TextOptionsFormat.FXBack.BlueColor = 0
 TextOptionsFormat.FXBack.Alpha = 75

 Options.UpButton.AnimTemplate = ATUpButton
 Options.UpButton.XPos = 455
 Options.UpButton.YPos = 140
 Options.DownButton.AnimTemplate = ATDownButton
 Options.DownButton.XPos = 455
 Options.DownButton.YPos = 182
 Options.CancelButton.AnimTemplate = ATXButton
 Options.CancelButton.XPos = 470
 Options.CancelButton.YPos = 75
End
```

B.9.8. Sección [MainMenu].

En esta sección se establecerá la configuración del menú principal. El menú principal será el que aparecerá nada más arrancar el motor y desde el que se podrá comenzar una nueva partida, recuperar una previamente salvada, ver los créditos y abandonar el juego para retornar a Windows.

Captura recortada donde se ven las opciones de un menú principal

En primer lugar se esperará leer la configuración de la imagen de fondo. Con la variable `BackImage.FileName` se indicará el nombre de la imagen de fondo. Asociada a la imagen de fondo se permtirá establecer animaciones estáticas, esto es, animaciones que se ejecuten en una determinada posición. El número de imágenes máximo permitido será de 256 y los valores asociar serán los del identificador de la plantilla de animación y posición a través de las variables `BackImage.Anim[x].AnimTemplate`, `BackImage.Anim[x].XPos` y `BackImage.Anim[x].YPos`. Siendo x un identificador entre 0 y 255 que indicará la animación concreta que se está estableciendo.

Asociado a esta interfaz podrá establecerse una música de fondo MIDI si bien, no será obligatorio. En caso de querer establecerse, se deberán de rellenar dos variables. Por un lado `MIDIMusic.FileName` para indicar el nombre del archivo MIDI y por otro lado `MIDIMusic.LoopFlag` para indicar si hay que reproducir cíclicamente (valor `true`) o no (valor `false`). Por defecto valdrá `false` con lo que una vez que acabe de reproducirse el fichero MIDI cesará toda la música.

Acto seguido se leerá la configuración relativa a los botones de opciones. Esta configuración estará orientada, como siempre en los botones, a leer el identificador de la plantillad e animación y la posición en la que éstos, los botones, serán dibujados. La configuración del botón de comenzar una nueva partida se leerá de las variables con prefijo `Options.StartNewGameButton.`, la de cargar una partida previamente guardada con `Options.LoadGameButton.`, la de ver los créditos de juego con `Options.CreditsButton.` y la de abandonar el juego con `Options.ExitButton.`. En todos los casos, se deberá de añadir `XPos` e `YPos` para especificar la posición del botón y `AnimTemplate` para indicar la plantilla de animación que lo representará. Acompañando la información relativa a la visión de los créditos, se deberá de suministrar también el identificador del perfil de presentación que se mostrará cuando se pulse esta opción a través de la variable `Options.CreditsButton.PresentationProfile`.

El siguiente apartado dependerá de si se ha rellenado información en el archivo de texto estático relativo al texto informativo que podrá aparecer en este interfaz. Las variables `VersionInfo.XPos` y `VersionInfo.YPos` situarian dicho texto mientras que el color RGB se establecería a través de `VersionInfo.RedColor`, `VersionInfo.GreenColor` y `VersionInfo.BlueColor`.

El último apartado estará destinado a la información de fade. Primeramente se establecerá el color que tendrá la pantalla nada más iniciar la interfaz a través de las variables `Fade.Start.RedColor`, `Fade.Start.GreenColor` y `Fade.Start.BlueColor`. La velocidad con la que se pasará de dicho color a los datos visuales de la interfaz se localizará con la variable `Fade.In.Speed`. Su valor deberá de ser superior a 0 (y estar situado en torno a 128 y 192 para obtener unas tasas aceptables). Cuando se desee abandonar la aplicación se realizará un fade desde el aspecto normal de la interfaz a un color, ese color RGB se establecerá en las variables `Fade.Out.RedColor`, `Fade.Out.GreenColor` y `Fade.Out.BlueColor` mientras que la velocidad a la que se hará se localizará en

`Fade.Out.Speed` teniendo en cuenta los mismos factores explicados para la variable `Fade.In.Speed`.

Ejemplo:

```
[MainMenu]
 BackImage.FileName = Graphics\Interfaz\MainMenuBackImg.tga

 MIDIMusic.FileName = midi\lands_02.mid
 MIDIMusic.LoopFlag = true

 Options.StartNewGameButton.AnimTemplate = ATNewGameOption
 Options.StartNewGameButton.XPos = 20
 Options.StartNewGameButton.YPos = 320
 Options.LoadGameButton.AnimTemplate = ATLoadGameOption
 Options.LoadGameButton.XPos = 40
 Options.LoadGameButton.YPos = 368
 Options.CreditsButton.AnimTemplate = ATCreditsOption
 Options.CreditsButton.XPos = 344
 Options.CreditsButton.YPos = 368
 Options.CreditsButton.PresentationProfile = Creditos
 Options.ExitButton.AnimTemplate = ATExitOption
 Options.ExitButton.XPos = 200
 Options.ExitButton.YPos = 415

 VersionInfo.XPos = 10
 VersionInfo.YPos = 463
 VersionInfo.RedColor = 255
 VersionInfo.GreenColor = 255
 VersionInfo.BlueColor = 255

 Fade.Start.RedColor = 0
 Fade.Start.GreenColor = 0
 Fade.Start.BlueColor = 0
 Fade.In.Speed = 196
 Fade.Out.RedColor = 0
 Fade.Out.GreenColor = 0
 Fade.Out.BlueColor = 0
 Fade.Out.Speed = 150
End
```

B.9.9. Sección [LoaderSavedFileManager].

En esta sección se establecerá la configuración de la interfaz que aparecerá cuando se desee recuperar una partida guardada. El acceso a esta interfaz podrá realizarse desde el menú principal o bien desde el menú de juego (el que sale al pulsar la tecla `ESC` durante el juego).

A grandes rasgos, el interfaz se compondrá de una imagen de fondo, con la posibilidad de asociar animaciones, un menú con los nombres de las partidas guardadas y una zona donde mostrar la descripción general de dichos archivos (relativos al nombre de área, hora en la que se grabó, nombre del jugador, etc) y ya por último los botones de navegación por la lista de partidas guardadas y los referidos a cargar una partida seleccionada y abandonar el interfaz. La posibilidad de borrar una partida sólo estará en la interfaz de partidas salvadas.

En primer lugar se esperará leer la configuración de la imagen de fondo. Con la variable `BackImage.FileName` se indicará el nombre de la imagen. Asociada a la imagen de fondo se permtirá establecer animaciones estáticas, esto es, que se ejecuten en una determinada posición. El número de imágenes máximo permitido será de 256 y los valores a asociar serán los del identificador de la plantilla de animación y posición a través de las variables `BackImage.Anim[x].AnimTemplate`, `BackImage.Anim[x].XPos` y `BackImage.Anim[x].YPos`. Siendo x un identificador entre 0 y 255 que indicará la animación concreta que se está estableciendo.

A continuación se deberá de configurar el espacio destinado a mostrar la lista de ficheros guardados. No se mostrará el nombre de los ficheros guardados sino una pequeñá descripción

que se pidió al guardar la partida. Lo primero que se leerá será el número máximo de archivos visibles en pantalla a través de la variable `SavedFilesList.NumberOfVisibleFiles`. Se deberá de introducir obligatoriamente un valor superior a 1. Despues se asociará la posición en donde comenzarán a mostrarse los nombres, siendo recogida a través de las variables `SavedFilesList.XPos` y `SavedFilesList.YPos`. Para la selección de una partida guardada se utilizará un sistema similar al del interfaz de selección de texto, esto es, usando un color distinto asociado a la pequeña descripción dependiendo de si ésta se seleccionada o no. Cuando sea seleccionada se usará el color establecido en las variables con prefijo `SavedFilesList.File.Name.Select.` mientras que si no es seleccionada, el color RGB utilizado será el que se haya establecido en las variables con prefijo `SavedFilesList.File.Name.Unselect.`. En todos los casos, éstos prefijos irán seguidos de las terminaciones `RedColor`, `GreenColor` y `BlueColor` para terminar de formar el nombre de las variables en las que especificar el color.

El último paso dentro de lo que es la configuración de la zona asociada a mostrar las partidas guardadas, será el de configurar el FXBack que se asociará a aquella descripción que resulte seleccionada. Para ello, se deberán de configurar las variables con prefijo `SavedFilesList.File.FXBack.Select.` seguidas de `RedColor`, `GreenColor` y `BlueColor` respectivamente, con el fin de indicar el color y `Alpha` para establecer el grado de transparencia del FXBack.

Cuando una partida sea guardada, el motor asociará automáticamente cierta información de manera automática. Esa información corresponderá al nombre del área en donde se encontraba el jugador, el propio nombre del personaje controlado por el jugador, su salud y la hora en la que sucedió la grabación de la partida. Para mostrar esa información será necesario indicar tanto la posición como el color RGB utilizado. Para todos los casos, habrá que especificar la misma información por lo que únicamente cambiará el prefijo de las variables. Así, en el caso del nombre del área se recogerá a través de las variables con prefijo `SelectSavedFileInfo.AreaName.`, en el caso del nombre del jugador a través de `SelectSavedFileInfo.PlayerName`, para la salud del jugador se utilizarán las variables `SelectSavedFileInfo.PlayerHealth.` y por último, indicaremos la posición y el color del texto con la hora a la que sucedió la grabación utilizando el prefijo `SelectSavedFileInfo.WorldTime.`. En todos los casos, se utilizarán las terminaciones `XPos` e `YPos` para indicar la posición y `RedColor`, `GreenColor` y `BlueColor` para el color.

Los botones de opción requerirán recibir el identificador de la plantilla de animación que los representa así como las coordenadas en donde comenzarán a dibujarse. Así, la configuración asociada al botón desplazar hacia arriba descripciones base de los archivos de partidas guardadas se recogerá en las variables con prefijo `Options.UpButton.`, las relativa a hacerlo hacia abajo con `Options.DownButton.`, en el caso de la carga de una partida con `Options.LoadButton.` y para la opción de salida `Options.ExitButton.` En todos los casos, se utilizarán las terminaciones `AnimTemplate`, `XPos` e `YPos` para la plantilla de animación y posiciones respectivamente.

El último apartado estará destinado a definir la información de fade. Por un lado tendremos la que será utilizada para pasar de la pantalla de interfaz a un color siempre y cuando se haya elegido una partida para cargar. Para establecer el color se utilizarán las variables `Fade.Out.RedColor`, `Fade.Out.GreenColor` y `Fade.Out.BlueColor`. Para asociar la velocidad a la que ocurrirá dicho fade utilizaremos la variable `Fade.Out.Speed`. Cuando accedamos a este interfaz desde el menú principal, en primer lugar se generará un fundido por lo que necesitaremos establecer aquí la velocidad a la que pasaremos de ese fundido a la pantalla completa que compone este interfaz. Lo haremos usando la variable `Fade.In.Speed`.

Ejemplo:

```
[LoaderSavedFileManager]
 BackImage.FileName = Graphics\Interfaz\LoaderBackImg.tga

 SavedFilesList.NumberOfVisibleFiles = 10
 SavedFilesList.XPos = 47
 SavedFilesList.YPos = 85
```

```
 SavedFilesList.File.FXBack.Select.RedColor = 0
 SavedFilesList.File.FXBack.Select.GreenColor = 155
 SavedFilesList.File.FXBack.Select.BlueColor = 155
 SavedFilesList.File.FXBack.Select.Alpha = 100
 SavedFilesList.File.Name.Select.RedColor = 255
 SavedFilesList.File.Name.Select.GreenColor = 255
 SavedFilesList.File.Name.Select.BlueColor = 255
 SavedFilesList.File.Name.Unselect.RedColor = 180
 SavedFilesList.File.Name.Unselect.GreenColor = 180
 SavedFilesList.File.Name.Unselect.BlueColor = 255

 SelectSavedFileInfo.AreaName.XPos = 329
 SelectSavedFileInfo.AreaName.YPos = 93
 SelectSavedFileInfo.AreaName.RedColor = 255
 SelectSavedFileInfo.AreaName.GreenColor = 255
 SelectSavedFileInfo.AreaName.BlueColor = 255
 SelectSavedFileInfo.PlayerName.XPos = 329
 SelectSavedFileInfo.PlayerName.YPos = 109
 SelectSavedFileInfo.PlayerName.RedColor = 255
 SelectSavedFileInfo.PlayerName.GreenColor = 200
 SelectSavedFileInfo.PlayerName.BlueColor = 0
 SelectSavedFileInfo.PlayerHealth.XPos = 329
 SelectSavedFileInfo.PlayerHealth.YPos = 125
 SelectSavedFileInfo.PlayerHealth.RedColor = 255
 SelectSavedFileInfo.PlayerHealth.GreenColor = 200
 SelectSavedFileInfo.PlayerHealth.BlueColor = 0
 SelectSavedFileInfo.WorldTime.XPos = 329
 SelectSavedFileInfo.WorldTime.YPos = 141
 SelectSavedFileInfo.WorldTime.RedColor = 255
 SelectSavedFileInfo.WorldTime.GreenColor = 255
 SelectSavedFileInfo.WorldTime.BlueColor = 0

 Options.UpButton.AnimTemplate = ATUpButton
 Options.UpButton.XPos = 300
 Options.UpButton.YPos = 300
 Options.DownButton.AnimTemplate = ATDownButton
 Options.DownButton.XPos = 300
 Options.DownButton.YPos = 355
 Options.LoadButton.AnimTemplate = ATLoaderLoadOption
 Options.LoadButton.XPos = 500
 Options.LoadButton.YPos = 400
 Options.ExitButton.AnimTemplate = ATSaverLoaderExitOption
 Options.ExitButton.XPos = 500
 Options.ExitButton.YPos = 440

 Fade.Out.RedColor = 0
 Fade.Out.GreenColor = 0
 Fade.Out.BlueColor = 0
 Fade.Out.Speed = 192
 Fade.In.Speed = 192
End
```

B.9.10. Sección [SaverSavedFileManager].

En esta opción se configurará el interfaz asociado al almacenamiento de partidas si bien, desde él también se podrán eliminar partidas guardadas con el fin de sanear el espacio en disco pues a medida que se vayan guardando partidas se ocupará espacio. También conviene tener en cuenta, que las partidas ocuparán más cuanto más áreas distintas se hayan visitado, esto es así porque al visitar un área se deberá de guardar en disco la situación en la que el jugador deja dicho área al visitarla. Todas las partidas guardadas irán a un directorio llamado `save\` que se creará si al arrancar el juego no existe.

La estructura de esta interfaz será idéntica a la usada para definir la destinada a la carga de archivos. La única diferencia consistirá en que se deberá de indicar un set distinto de botones de opción así como un apartado para configurar la zona donde se introducirá la descripción del archivo a guardar. Lo que no existirá será un campo destinado a configurar fades pues a esta

interfaz se podrá acceder sólo desde el menú de juego. Cuando hagamos tanto un borrado como una grabación, el motor permanecerá en esta interfaz. Al abandonarla el retorno al menú de juego será, además, inmediato.

En primer lugar se esperará leer la configuración de la imagen de fondo. Con la variable `BackImage.FileName` se indicará el nombre de la imagen de fondo. Asociada a la imagen de fondo se permtirá establecer animaciones estáticas, esto es, que se ejecutan en una determinada posición. El número de imágenes máximo permitido será de 256 y los valores asociar serán los del identificador de la plantilla de animación y posición a través de las variables `BackImage.Anim[x].AnimTemplate`, `BackImage.Anim[x].XPos` y `BackImage.Anim[x].YPos`. Siendo x un identificador entre 0 y 255 que indicará la animación concreta que se está estableciendo.

A continuación se deberá de configurar el espacio destinado a mostrar la lista de ficheros guardados. No se mostrará el nombre de los ficheros guardados sino una pequeñá descripción que se pidió al guardar la partida. Lo primero que se leerá será el número máximo de archivos visibles en pantalla a través de la variable `SavedFilesList.NumberOfVisibleFiles`. Se deberá de introducir obligatoriamente un valor superior a 1. Despues se asociará la posición en donde comenzarán a mostrarse los nombres, siendo recogida dicha información a través de las variables `SavedFilesList.XPos` y `SavedFilesList.YPos`. Para la selección de una partida guardada se utilizará un sistema similar al del interfaz de selección de texto, esto es, usando un color distinto asociado a la pequeña descripción dependiendo de si ésta se seleccionada o no. Cuando sea seleccionada se usará el color establecido en las variables con prefijo `SavedFilesList.File.Name.Select.` mientras que cuando no lo sea se utilizará el prefijo `SavedFilesList.File.Name.Unselect.`. Los nombres de las variables se completarán con `RedColor`, `GreenColor` y `BlueColor` respectivamente.

El último paso dentro de lo que es la configuración de la zona asociada a mostrar las partidas guardadas, será el de configurar el FXBack que se asociará a aquella descripción que resulte seleccionada. Para ello, se deberán de configurar las variables con prefijo `SavedFilesList.File.FXBack.Select.` que servirán para definir los valores de color y alpha del FXBack. Así, mediante las terminaciones `RedColor`, `GreenColor` y `BlueColor` se especificará el color y con `Alpha` se hará lo propio con el componente de transparencia.

Cuando una partida sea guardada, el motor asociará automáticamente cierta información de manera automática. Esa información corresponderá al nombre del área en donde se encontraba el jugador, el propio nombre del personaje controlado por el jugador, su salud y la hora en la que sucedió la grabación de la partida. Para mostrar esa información será necesario indicar tanto la posición como el color RGB utilizado. Para todos los casos, habrá que especificar la misma información por lo que únicamente cambiará el prefijo de las variables. Así, en el caso del nombre del área se recogerá a través de las variables con prefijo `SelectSavedFileInfo.AreaName.`, en el caso del nombre del jugador a través de `SelectSavedFileInfo.PlayerName`, para la salud del jugador se utilizarán las variables `SelectSavedFileInfo.PlayerHealth.` y por último, indicaremos la posición y el color del texto con la hora a la que sucedió la grabación utilizando el prefijo `SelectSavedFileInfo.WorldTime.`. En todos los casos, se utilizarán las terminaciones `XPos` e `YPos` para indicar la posición y `RedColor`, `GreenColor` y `BlueColor` para el color.

El siguiente apartado será el destinado a configurar la forma en la que se introducirán las descripciones de las partidas guardadas. Tendrá dos partes. Por un lado se esperará leer la configuración asociada a un texto estático relativo a indicar que se espera recibir una descripción. En las variables con prefijo `SavedFileDesc.InfoText.` y terminación `XPos` e `YPos` se indicará la posición en donde deberá de escribir, mientras que las variables para establecer el color RGB serán aquellas de prefijo `SavedFileDesc.InfoText.` y terminación `RedColor`, `BlueColor` y `GreenColor` respectivamente. La segunda parte estará destinada a especificar la forma en la que dicha descripción se recogerá. Todas las variables de este apartado comenzarán con el prefijo `SavedFileDesc.Input.`. Así, con la terminación `MaxCharacters` indicaremos el número máximo de caracteres que podrá tener la descripción. El valor que se asocie deberá de ser, necesariamente, superior a uno. El color del texto con el

que se recogerá se hará a través de las terminaciones `RedColor`, `GreenColor` y `BlueColor`. Las dos últimas variables, formadas con las terminaciones `XPos` e `YPos`, estarán destinada a indicar la posición en donde se localizará el lugar en donde serán introducirdos los caracteres.

Los botones de opción requerirán recibir el identificador de la plantilla de animación que los representa así como las coordenadas en donde comenzarán a dibujarse. Todos se configurarán en los mismos términos, cambiando sólo los prefijos de las variables. Así, la configuración asociada al botón desplazar hacia arriba descripciones base de los archivos de partidas guardadas se recogerá con el prefijo `Options.UpButton.AnimTemplate`, la de desplazar hacia abajo con `Options.DownButton.`, la opción de guardar la partida con `Options.SaveButton.`, el botón para borrar la partida actualmente seleccionada usará el prefijo `Options.RemoveButton.` y por por último, la de abandonar la interfaz con `Options.ExitButton.`. Para todos los prefijos, se usarán las terminaciones `AnimTemplate`, `XPos` e `YPos` con el fin de indicar la plantilla de animación asociada y la posición respectivamente.

Ejemplo:

```
[SaverSavedFileManager]
 BackImage.FileName = Graphics\Interfaz\SaverBackImg.tga

 SavedFilesList.NumberOfVisibleFiles = 10
 SavedFilesList.XPos = 47
 SavedFilesList.Ypos = 85
 SavedFilesList.File.FXBack.Select.RedColor = 0
 SavedFilesList.File.FXBack.Select.GreenColor = 155
 SavedFilesList.File.FXBack.Select.BlueColor = 155
 SavedFilesList.File.FXBack.Select.Alpha = 100
 SavedFilesList.File.Name.Select.RedColor = 255
 SavedFilesList.File.Name.Select.GreenColor = 255
 SavedFilesList.File.Name.Select.BlueColor = 255
 SavedFilesList.File.Name.Unselect.RedColor = 180
 SavedFilesList.File.Name.Unselect.GreenColor = 180
 SavedFilesList.File.Name.Unselect.BlueColor = 255

 SelectSavedFileInfo.AreaName.XPos = 329
 SelectSavedFileInfo.AreaName.YPos = 93
 SelectSavedFileInfo.AreaName.RedColor = 255
 SelectSavedFileInfo.AreaName.GreenColor = 255
 SelectSavedFileInfo.AreaName.BlueColor = 255
 SelectSavedFileInfo.PlayerName.XPos = 329
 SelectSavedFileInfo.PlayerName.YPos = 109
 SelectSavedFileInfo.PlayerName.RedColor = 255
 SelectSavedFileInfo.PlayerName.GreenColor = 200
 SelectSavedFileInfo.PlayerName.BlueColor = 0
 SelectSavedFileInfo.PlayerHealth.XPos = 329
 SelectSavedFileInfo.PlayerHealth.YPos = 125
 SelectSavedFileInfo.PlayerHealth.RedColor = 255
 SelectSavedFileInfo.PlayerHealth.GreenColor = 200
 SelectSavedFileInfo.PlayerHealth.BlueColor = 0
 SelectSavedFileInfo.WorldTime.XPos = 329
 SelectSavedFileInfo.WorldTime.YPos = 141
 SelectSavedFileInfo.WorldTime.RedColor = 255
 SelectSavedFileInfo.WorldTime.GreenColor = 255
 SelectSavedFileInfo.WorldTime.BlueColor = 0

 SavedFileDesc.InfoText.XPos = 326
 SavedFileDesc.InfoText.YPos = 200
 SavedFileDesc.InfoText.RedColor = 255
 SavedFileDesc.InfoText.BlueColor = 255
 SavedFileDesc.InfoText.GreenColor = 255
 SavedFileDesc.Input.MaxCharacters = 20
 SavedFileDesc.Input.RedColor = 255
 SavedFileDesc.Input.GreenColor = 200
 SavedFileDesc.Input.BlueColor = 0
 SavedFileDesc.Input.XPos = 326
```

```
 SavedFileDesc.Input.YPos = 223

 Options.UpButton.AnimTemplate = ATUpButton
 Options.UpButton.XPos = 300
 Options.UpButton.YPos = 300
 Options.DownButton.AnimTemplate = ATDownButton
 Options.DownButton.XPos = 300
 Options.DownButton.YPos = 355
 Options.SaveButton.AnimTemplate = ATSaverSaveOption
 Options.SaveButton.XPos = 500
 Options.SaveButton.YPos = 360
 Options.RemoveButton.AnimTemplate = ATSaverDeleteOption
 Options.RemoveButton.XPos = 500
 Options.RemoveButton.YPos = 400
 Options.ExitButton.AnimTemplate = ATSaverLoaderExitOption
 Options.ExitButton.XPos = 500
 Options.ExitButton.YPos = 440
End
```

B.9.11. Sección [PlayerCreator].

Esta será la interfaz que se active cuando desde el menú principal se escoja comenzar una nueva partida. Además de trabajar de forma intensiva con el archivo de reglas, esta interfaz utilizará como punto de apoyo el contenido definido en el archivo de perfiles de jugador pues será aquí desde donde el usuario pueda crear o editar al héroe o heroína que le represente en el juego.

Ejemplo de pantalla de creación de personaje

En primer lugar, se esperará recibir el color del texto flotante. El texto flotante será el que aparecerá a modo de ayuda cuando coloquemos el cursor sobre una determinada posición. Por ejemplo, cuando seleccionemos un atributo del jugador, el texto flotante aparecerá para indicarnos la cantidad de puntos de distribución que nos costará aumentar en una unidad el valor actual del jugador que estamos creando para dicho atributo. El color RGB del texto flotante ser recogerá a través de las variables `FloatingText.RedColor`, `FloatingText.GreenColor` y `FloatingText.BlueColor`.

Como en todas las grandes interfaces, ésta deberá de tener asociada una imagen de fondo que ocupe toda la pantalla. Con la variable `BackImage.FileName` se indicará el nombre de la imagen de fondo. Asociado a la imagen de fondo se permitirá establecer animaciones estáticas, esto es, que se ejecuten en una determinada posición. El número de imágenes máximo permitido será de 256 y los valores asociar serán los del identificador de la plantilla de animación y posición a través de las variables `BackImage.Anim[x].AnimTemplate`, `BackImage.Anim[x].XPos` y `BackImage.Anim[x].YPos`. Siendo x un identificador entre 0 y 255 que indicará la animación concreta que se está estableciendo.

Asociado a esta interfaz podrá establecerse una música de fondo MIDI si bien, no será obligatorio. En caso de querer establecerse, se deberán de rellenar dos variables. Por un lado `MIDIMusic.FileName` para indicar el nombre del archivo MIDI y por otro lado `MIDIMusic.LoopFlag` para indicar si hay que reproducir cíclicamente (valor `true`) o no (valor `false`). Por defecto valdrá `false` con lo que una vez que acabe de reproducirse el fichero MIDI cesará toda la música.

El siguiente paso será establecer los botones que representen las dos opciones generales de la interfaz. Estas opciones generales serán las de cancelar el proceso de creación del jugador con retorno al menú principal y la de aceptar al jugador con comienzo del juego. La opción de aceptar se activará sólo si se cumplen las condiciones mínimas para ello. Estas condiciones mínimas serán las de haber introducido el nombre del jugador y haber repartido todos los puntos, si así fue especificado en el archivo de perfil para jugador. Mediante las variables con prefijo `Options.CancelButton.` se configurará el botón de cancelación, mientras que aquellas que poseean el prefijo `Options.AcceptButton.` harán los propio con el botón de aceptación. En todos los casos, las terminaciones `AnimTemplate`, `XPos` e `YPos` servirán para especificar la plantilla de animación y la posición respectivamente.

El siguiente paso será configurar los campos relativos a la introducción del nombre. Por un lado se configurará el texto estático que servirá para poder introducir el nombre, pues al seleccionarlo y pulsar al botón izquierdo se pedirá explícitamente el nombre del jugador. Las variables con prefijo `Name.InfoText.` especificarán la configuración relativa a la información del nombre. Así, las terminaciones `XPos` e `YPos` establecerán la posición, mientras que el color cuando el texto esté seleccionado se indicará con las terminaciones `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor`. Para el texto sin selección, se utilizarán las terminaciones `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor`.

Una vez configurado el texto indicativo se hará lo propio con el apartado para introducir los datos. En este caso, los prefijos de las variables vendrán definidos por `Name.Input.`. Así, con las terminaciones `XPos` e `YPos` indicaremos la posición en donde se comenzará a escribir el nombre, mientras que la terminación `MaxCharacters` servirá para indicar el número máximo de caracteres que el nombre podrá tener. Lógicamente, su valor deberá de ser superior a 0. Cuando se seleccione el texto indicativo para la introducción del nombre, ya explicado anteriormente, se hará lo propio con el texto del nombre actualmente introducido. Así pues, en las terminaciones `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor` se indicará ese color. De la misma forma, cuando no se esté introduciendo un nombre o el texto indicativo no se halle seleccionado, el color utilizado estará definido en las terminaciones `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor`.

Una vez configurado el apartado destinado al establecimiento del nombre, vendrá la configuración del apartado donde se escogerá el retrato. Mediante las terminaciones `PlayerPortrait.XPos` y `PlayerPortrait.YPos` se indicará la posición en donde será dibujado el retrato. El avance y retroceso por los retratos se hará utilizando botones de avance y retroceso. En el caso del botón de avance de retratos tendremos los prefijos PlayerPortrait`.Options.NextButton.` y en el caso del botón de retroceso de retratos `PlayerPortrait.Options.PrevButton.`. En ambos casos, se utilizarán las terminaciones `AnimTemplate`, `XPos` e `YPos` para indicar plantilla de animacion y posición respectivamente. Habrá que tener en cuenta que los retratos se habrán configurado, en el archivo de perfiles de jugador, en torno a los géneros así que al cambiar de género se cambiará de forma automática todo el conjunto de retratos que podremos elegir.

El siguiente paso será configurar todo lo relativo a la elección del género, del tipo y de la clase de la criatura que queremos elegir. En los tres casos, lo que se va a esperar leer en las variables correspondientes va a ser lo mismo. Por un lado habrá que configurar la posición donde se situará un texto estático cuya finalidad será la de indicar el cambio de género, tipo y clase respectivamente. Ese texto estático podrá estar seleccionado y sin seleccionar y en ambos casos se esperará que el color sea diferente.

Por otro lado, habrá que configurar todo lo concerniente a indicar qué género, tipo y clase se haya establecido. Para ello únicamente necesitaremos indicar la posición y el color de dicho texto. Además, para ayudar a indicar una selección se esperará que sea configurado un FXBack que se dibujará por debajo del género, tipo y clase que se halle establecido en un determinado momento.

Dicho lo anterior, la configuración del género en lo relativo al texto estático indicativo se realizará a través de las variables con prefijo `Genre.InfoText.`, usándose las terminaciones `XPos` e `YPos` para la posición, `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor` para el color del texto cuando se halle seleccionado y `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor` para el color cuando no se halle seleccionado. En lo relativo a la configuración del género seleccionado usaremos las variables con prefijo `Genre.Selection.` y terminaciones `XPos` e `YPos` para indicar la posición, `RedColor`, `GreenColor` y `BlueColor` para el color RGB y, por último, `FXBack.RedColor`, `FXBack.GreenColor`, `FXBack.BlueColor` y `Alpha` para hacer lo propio con el color y el valor de transparencia del FXBack asociado.

La configuración del tipo en lo relativo al texto estático indicativo se realizará a través de las variables cuyo prefijo sea `Type.InfoText.`. De esta forma, las terminaciones `XPos` e `YPos` se usarán para la posición, `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor`, para el color del texto cuando se halle seleccionado y `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor` para el color cuando no lo esté. En lo relativo a la configuración del tipo seleccionado usaremos las variables con prefijo `Type.Selection.` y terminaciones `XPos` e `YPos` para indicar la posición, `RedColor`, `GreenColor` y `BlueColor` para el color RGB y, por último, configuraremos el FXBack usando `FXBack.RedColor`, `FXBack.GreenColor` y `FXBack.BlueColor` para el color RGB y `FXBack.Alpha` para hacer lo propio con el valor de transparencia. Habrá que tener en cuenta que cuando cambiemos de tipo automáticamente el conjunto de clases posibles también cambiará (repasar la edición con el archivo de perfiles de jugador).

En el caso de la configuración de la clase en lo relativo al texto estático indicativo se realizará a través de las variables cuyo prefijo sea `Class.InfoText.`. De esta forma, las terminaciones `XPos` e `YPos` se usarán para la posición, `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor` para el color del texto cuando se halle seleccionado y `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor` para el color cuando no se halle seleccionado. En lo relativo a la configuración de la clase seleccionada usaremos las variables con prefijo `Class.Selection.` y terminación `XPos` e `YPos` para indicar la posición, `RedColor`, `GreenColor` y `BlueColor` para el color RGB y, por último, configuraremos el FXBack usando `FXBack.RedColor`, `FXBack.GreenColor` y `FXBack.BlueColor` para asociar el color RGB y `FXBack.Alpha` para hacer lo propio con el valor de transparencia.

El siguiente paso consistirá configurar un gran bloque destinado al trabajo con los atributos. Los atributos comprenderán la muestra del nombre de todos los atributos que dispondrá la criatura que estamos editados junto a su valor base actual y una barra de porcentaje que indicará el total de puntos de distribución que tiene el usuario para mejorar sus atributos. Además, todo atributo seleccionado para incrementar su valor o quitar puntos de distribución que fueron añadidos, dispondrá de un FXBack que aclarará aún más su selección. El modo de asociar o quitar puntos a los atributos (sólo se podrán quitar puntos si previamente fueron asociados) será utilizando dos botones de opciones, uno de incremento y otro de decremento.

En primer lugar, para la configuración de los atributos, estableceremos el formato con el que serán dibujados los elementos. Por elementos entendermos el nombre, los valores base, los de incremento y los puntos de distribución. La configuación del nombre ser hará utilizando el prefijo `Attributes.DrawFormat.Names.`. A este prefijo se le añadirán las terminaciones `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor` para especificar el color cuando sea seleccionado y `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor` para hacer lo propio con la deselección. La configuración de los

valores base poseerán el prefijo Attributes.DrawFormat.BaseValues., teniéndose las terminaciones RedColor, GreenColor y BlueColor para especificar el color, mientras que la configuración del FXBack asociado al atributo seleccionado será leída a través de las terminaciones FXBack.RedColor, FXBack.GreenColor, FXBack.BlueColor y FXBack.Alpha para el color y el valor de transparencia respectivamente. El color de los valores base cuando estos poseean algún tipo de punto de distribución asociado, vendrá regido por las variables de prefijo Attributes.DrawFormat.ImproveValues. y terminación RedColor, GreenColor y BlueColor. Por último, el color de los puntos de distribución que quedan se podrá establecer usando las variables con prefijo Attributes.DrawFormat.DistributionPoints. y terminación RedColor, GreenColor y BlueColor.

Una vez establecido las configuraciones base de dibujado anterior, se pasará a localizar en pantalla los diferentes componentes que componen el bloque de configuración para los atributos. En primer lugar, se configurará la barra de porcentaje utiliznado el prefijo Attributes.DistributionPoints.PercentageBar.. Mediante las terminaciones XPos e YPos indicaremos su posición de dibujado, con Lenght y Width su longitud y anchura en pixels. En ambos casos, los valores deberán de oscilar entre 1 y 256. Con la terminación HorizontalDrawFlag indicaremos si queremos que la barra se dibuje en horizontal (true) o en vertical (false). Y, por último, usando la terminación Alpha estableceramos el valor de transparencia asociado a la misma. En sintonía con la barra de porcentaje estará la localización de los puntos de pocentaje, esto es, el ofrecer la misma información que la barra de porcentaje pero de manera numérica. Con las variables cuyo prefijo sean Attributes.DistributionPoints y terminación XPos e YPos podremos indicar la posición donde mostrar esta información.

El siguiente paso en la configuración de los atributos, será indicar la posición donde se irán a poner, precisamente, los atributos en cuanto a su nombre y su valor base. En las variables con prefijo Attributes.Health.Name. y terminación XPos e YPos indicaremos donde se colocará el nombre del atributo salud (que se habrá definido como texto estático, al igual que el resto de atributos a configurar), mientras que usando el prefijo Attributes.Health.BaseValue. y las terminaciones XPos e YPos dónde aparecerá el valor base de dicho atributo. Con las variables cuyo prefijo sea Attributes.CombatPoints.Name. y terminación XPos e YPos haremos lo propio con el nombre del atributo puntos de combate mientras que usando el prefijo Attributes.CombatPoints.BaseValue. y las terminaciones XPos e YPos localizaremos la posición en donde se mostrará el valor base de dichos puntos de combate. Para el trabajo con los atributos extendidos, usaremos las variables con prefijo Attributes.ExtendedAttribute[x].. De esta forma, las terminaciones Name.XPos y Name.YPos nos servirán para indicar la posición del nombre del atributo extendido, BaseValue.XPos y BaseValue.YPos harán lo propio con el valor base, teniendo en cuenta que la x representará el identificador del atributo extendido que se habrá definido en el archivo de reglas.

Los dos últimos elementos a configurar en torno a los atributos extendidos serán los relativos a los botones de opciones de incremento y decremento. En ambos casos habrá que establecer la misma información. Mediante el prefijo Attributes.Options.IncrementValueButton. se establecerán las variables con la información de los valores de incremento y mediante el prefijo Attributes.Options.DecrementValueButton. se hará lo propio con las variables de decremento. Así, usando las terminaciones AnimTemplate, XPos e YPos especificaremos plantilla de animación y posiciones de los botones.

El siguiente gran paso será la configuración de todo lo relativo a las habilidades. En este caso, deberemos de tener en cuenta de que se tendrá que configurar la posición e iconos de las habilidades (los iconos serán plantillas de animación con la misma configuración que un botón, esto es, al menos dos estados de animación), su nombre y la barra de porcentaje con los puntos de distribución.

Al igual que ocurría con los atributos, habrá un apartado en donde se configurará el modo de dibujado general en lo relativo al color. Con las variables cuyo prefijo sea `Habilities.DrawFormat.Names.` se establecerá la configuración relativa a los nombres de las mismas. Así, utilizando las terminaciones `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor` tendremos las variables con las que configuraremos el color RGB de los nombres de habilidades seleccionadas por el cursor, mientras que con las terminaciones `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor` tendremos lo propio para cuando no estén seleccionadas. Para ayudar a indicar el establecimiento de una habilidad, se usará un FXBack que se activará de forma permanente siempre que una habilidad haya sido establecida y sólo desaparecerá si dicha habilidad se quita como establecida en el jugador. Para configurar el FXBack usaremos las terminaciones `FXBack.RedColor`, `FXBack.GreenColor` y `FXBack.BlueColor` para indicar el color RGB y `FXBack.Alpha` para el componente de transparencia. El último paso en esta configuración, será indicar el color de los puntos de distribución que quedan por repartir en las habilidades. Se usará el prefijo `Habilities.DrawFormat.DistributionPoints.` con las terminaciones `RedColor`, `GreenColor` y `BlueColor`.

Con los datos anteriores, pasaremos a configurar la forma en la que se mostrarán las habilidades propiamente dichas. Para ello, deberemos de ser conscientes de que una habilidad se indicará mediante su nombre y un icono. Este icono, como se dijo con anterioridad, cumplirá las mismas condiciones, en lo relativo a su plantilla de animación asociada, que la que cumplen los botones. Con las variables cuyo prefijo sea `Habilities.Hability[x].` podremos añadir las terminaciones `Icon.AnimTemplate`, `Icon.XPos` e `Icon.YPos` para formar las variables que configuren lo relativo al icono que las representa. Mientras que usando las terminaciones `Name.XPos` y `Name.YPos` haremos lo propio en cuanto a la posición del nombre que indique a esa habilidad. El valor de x hará referencia a un identificador de habilidad definido en el archivo de reglas.
El último paso en la configuración de las habilidades será el relativo a configurará la barra de porcentaje. A través de las variables cuyo prefijo sea `Habilities.DistributionPoints.PercentageBar.` podremos formar las variables que nos interesen. Así, añadiendo las terminaciones `XPos` e `YPos` indicaremos su posición de dibujado, con `Lenght` y `Width` su longitud y anchura en pixels (en ambos casos, los valores deberán de oscilar entre 1 y 256), con la terminación `HorizontalDrawFlag` indicaremos si queremos que la barra se dibuje en horizontal (`true`) o en vertical (`false`) y por último, usando la terminación `Alpha` estableceramos el valor de transparencia asociado a la misma. En sintonía con la barra de porcentaje estará la localización de los puntos de pocentaje, esto es, el ofrecer la misma información que la barra de porcentaje pero de manera numérica. Con las variables cuyo prefijo sea `Habilities.DistributionPoints.` y cuya termianción sea `XPos` e `YPos` podremos indicar la posición donde mostrar esta información.

Para cerrar la configuración de esta sección, deberemos de establecer los valores de fade. En primer se deberá de indicar un color que será el que se utilizará para realizar un fade desde dicho color a la imagen que constituye la interfaz propiamente dicha y, además, teniendo en cuenta de que dicho color se activará nada más establecer esta interfaz. Así pues, mediante las variables `Fade.Start.RedColor`, `Fade.Start.GreenColor` y `Fade.Start.BlueColor` asociaremos dicho color y con `Fade.In.Speed` indicaremos la velocidad a la que queremos que se realice el fade. Teniendo en cuenta que la velocidad deberá de ser superior a 0 y oscilar entre 128 y 192 para conseguir un fade suave.

Siempre que se pulsen a las opciones de cancelar y aceptar se activará un fade que irá desde el aspecto normal de la interfaz a un determinado color, antes de proceder a ejecutar el menú principal, si se dio a cancelar, o comenzar la carga del juego si se dio a aceptar. En todos los casos, trabajaremos con variables cuyo prefijo será `Fade.Out.`. Asi, para configurar el fade resultante de dar a cancelar dispondremos de las terminaciones `OnCancel.RedColor`, `OnCancel.GreenColor`, `OnCancel.BlueColor` y `OnCancel.Speed` mientras que para hacer lo propio con el fade asociado a la opción aceptar las variables a usar serán `OnAccept.RedColor`, `OnAccept.GreenColor`, `OnAccept.BlueColor` y `OnAccept.Speed`. Las precauciones en el establecimiento del valor usando las terminaciones

`.OnCancel.Speed` y `OnAccept.Speed` serán las mismas que las explicadas más atrás para `Fade.In.Speed`.

Ejemplo:

```
[PlayerCreator]
 FloatingText.RedColor = 255
 FloatingText.GreenColor = 255
 FloatingText.BlueColor = 255

 BackImage.FileName = Graphics\Interfaz\CharCreatorBackImg.tga

 MIDIMusic.FileName = midi\lands_33.mid
 MIDIMusic.LoopFlag = true

 Options.CancelButton.AnimTemplate = ATCancelButton
 Options.CancelButton.XPos = 20
 Options.CancelButton.YPos = 90
 Options.AcceptButton.AnimTemplate = ATStatueAcceptButton
 Options.AcceptButton.XPos = 560
 Options.AcceptButton.YPos = 340

 Name.InfoText.XPos = 265
 Name.InfoText.YPos = 190
 Name.InfoText.Select.RedColor = 255
 Name.InfoText.Select.GreenColor = 255
 Name.InfoText.Select.BlueColor = 255
 Name.InfoText.Unselect.RedColor = 180
 Name.InfoText.Unselect.GreenColor = 180
 Name.InfoText.Unselect.BlueColor = 225
 Name.Input.XPos = 315
 Name.Input.YPos = 190
 Name.Input.MaxCharacters = 20
 Name.Input.Select.RedColor = 255
 Name.Input.Select.GreenColor = 200
 Name.Input.Select.BlueColor = 0
 Name.Input.Unselect.RedColor = 200
 Name.Input.Unselect.GreenColor = 200
 Name.Input.Unselect.BlueColor = 0

 PlayerPortrait.XPos = 304
 PlayerPortrait.YPos = 120
 PlayerPortrait.Options.NextButton.AnimTemplate = ATRightButton
 PlayerPortrait.Options.NextButton.XPos = 380
 PlayerPortrait.Options.NextButton.YPos = 140
 PlayerPortrait.Options.PrevButton.AnimTemplate = ATLeftButton
 PlayerPortrait.Options.PrevButton.XPos = 260
 PlayerPortrait.Options.PrevButton.YPos = 140

 Genre.InfoText.XPos = 160
 Genre.InfoText.YPos = 380
 Genre.InfoText.Select.RedColor = 255
 Genre.InfoText.Select.GreenColor = 255
 Genre.InfoText.Select.BlueColor =  255
 Genre.InfoText.Unselect.RedColor = 180
 Genre.InfoText.Unselect.GreenColor = 180
 Genre.InfoText.Unselect.BlueColor = 225
 Genre.Selection.XPos = 210
 Genre.Selection.YPos = 380
 Genre.Selection.RedColor = 255
 Genre.Selection.GreenColor = 200
 Genre.Selection.BlueColor = 0
 Genre.Selection.FXBack.RedColor = 150
 Genre.Selection.FXBack.GreenColor = 50
 Genre.Selection.FXBack.BlueColor = 150
 Genre.Selection.FXBack.Alpha = 100

 Type.InfoText.XPos = 285
```

```
Type.InfoText.YPos = 380
Type.InfoText.Select.RedColor = 255
Type.InfoText.Select.GreenColor = 255
Type.InfoText.Select.BlueColor = 255
Type.InfoText.Unselect.RedColor = 180
Type.InfoText.Unselect.GreenColor = 180
Type.InfoText.Unselect.BlueColor = 225
Type.Selection.XPos = 335
Type.Selection.YPos = 380
Type.Selection.RedColor = 255
Type.Selection.GreenColor = 200
Type.Selection.BlueColor = 0
Type.Selection.FXBack.RedColor = 150
Type.Selection.FXBack.GreenColor = 50
Type.Selection.FXBack.BlueColor = 150
Type.Selection.FXBack.Alpha = 100

Class.InfoText.XPos = 405
Class.InfoText.YPos = 380
Class.InfoText.Select.RedColor = 255
Class.InfoText.Select.GreenColor = 255
Class.InfoText.Select.BlueColor =  255
Class.InfoText.Unselect.RedColor = 180
Class.InfoText.Unselect.GreenColor = 180
Class.InfoText.Unselect.BlueColor = 225
Class.Selection.XPos = 455
Class.Selection.YPos = 380
Class.Selection.RedColor = 255
Class.Selection.GreenColor = 200
Class.Selection.BlueColor = 0
Class.Selection.FXBack.RedColor = 150
Class.Selection.FXBack.GreenColor = 50
Class.Selection.FXBack.BlueColor = 150
Class.Selection.FXBack.Alpha = 100

Attributes.DrawFormat.Names.Select.RedColor = 255
Attributes.DrawFormat.Names.Select.GreenColor = 255
Attributes.DrawFormat.Names.Select.BlueColor = 255
Attributes.DrawFormat.Names.Unselect.RedColor = 180
Attributes.DrawFormat.Names.Unselect.GreenColor = 180
Attributes.DrawFormat.Names.Unselect.BlueColor =  200
Attributes.DrawFormat.BaseValues.RedColor = 255
Attributes.DrawFormat.BaseValues.GreenColor = 200
Attributes.DrawFormat.BaseValues.BlueColor = 0
Attributes.DrawFormat.BaseValues.FXBack.RedColor = 0
Attributes.DrawFormat.BaseValues.FXBack.GreenColor = 0
Attributes.DrawFormat.BaseValues.FXBack.BlueColor = 155
Attributes.DrawFormat.BaseValues.FXBack.Alpha = 100
Attributes.DrawFormat.ImproveValues.RedColor = 255
Attributes.DrawFormat.ImproveValues.GreenColor = 255
Attributes.DrawFormat.ImproveValues.BlueColor = 0
Attributes.DrawFormat.DistributionPoints.RedColor = 255
Attributes.DrawFormat.DistributionPoints.GreenColor = 255
Attributes.DrawFormat.DistributionPoints.BlueColor = 0

Attributes.DistributionPoints.XPos = 160
Attributes.DistributionPoints.YPos = 330
Attributes.DistributionPoints.PercentageBar.XPos = 155
Attributes.DistributionPoints.PercentageBar.YPos = 330
Attributes.DistributionPoints.PercentageBar.HorizontalDrawFlag = true
Attributes.DistributionPoints.PercentageBar.Lenght = 150
Attributes.DistributionPoints.PercentageBar.Width = 14
Attributes.DistributionPoints.PercentageBar.Alpha = 150

Attributes.Health.Name.XPos = 150
Attributes.Health.Name.YPos = 225
Attributes.Health.BaseValue.XPos = 260
Attributes.Health.BaseValue.YPos = 225
```

```
 Attributes.CombatPoints.Name.XPos = 150
 Attributes.CombatPoints.Name.YPos = 245
 Attributes.CombatPoints.BaseValue.XPos = 260
 Attributes.CombatPoints.BaseValue.YPos = 245
 Attributes.ExtendedAttribute[Destreza].Name.XPos = 150
 Attributes.ExtendedAttribute[Destreza].Name.YPos = 265
 Attributes.ExtendedAttribute[Destreza].BaseValue.XPos = 260
 Attributes.ExtendedAttribute[Destreza].BaseValue.YPos = 265

 Attributes.ExtendedAttribute[Fuerza].Name.XPos = 150
 Attributes.ExtendedAttribute[Fuerza].Name.YPos = 285
 Attributes.ExtendedAttribute[Fuerza].BaseValue.XPos = 260
 Attributes.ExtendedAttribute[Fuerza].BaseValue.YPos = 285

 Attributes.ExtendedAttribute[Suerte].Name.XPos = 150
 Attributes.ExtendedAttribute[Suerte].Name.YPos = 305
 Attributes.ExtendedAttribute[Suerte].BaseValue.XPos = 260
 Attributes.ExtendedAttribute[Suerte].BaseValue.YPos = 305

 Attributes.Options.IncrementValueButton.AnimTemplate = ATIncButton
 Attributes.Options.IncrementValueButton.XPos = 90
 Attributes.Options.IncrementValueButton.YPos = 245
 Attributes.Options.DecrementValueButton.AnimTemplate = ATDecButton
 Attributes.Options.DecrementValueButton.XPos = 90
 Attributes.Options.DecrementValueButton.YPos = 285

 Habilities.DrawFormat.Names.Select.RedColor = 255
 Habilities.DrawFormat.Names.Select.GreenColor = 255
 Habilities.DrawFormat.Names.Select.BlueColor = 255
 Habilities.DrawFormat.Names.Unselect.RedColor = 180
 Habilities.DrawFormat.Names.Unselect.GreenColor = 180
 Habilities.DrawFormat.Names.Unselect.BlueColor =  200
 Habilities.DrawFormat.Names.FXBack.RedColor = 225
 Habilities.DrawFormat.Names.FXBack.GreenColor = 0
 Habilities.DrawFormat.Names.FXBack.BlueColor = 0
 Habilities.DrawFormat.Names.FXBack.Alpha = 100
 Habilities.DrawFormat.DistributionPoints.RedColor = 255
 Habilities.DrawFormat.DistributionPoints.GreenColor = 255
 Habilities.DrawFormat.DistributionPoints.BlueColor = 0

 Habilities.Hability[Magia].Icon.AnimTemplate = ATHabilityMagic
 Habilities.Hability[Magia].Icon.XPos = 350
 Habilities.Hability[Magia].Icon.YPos = 225
 Habilities.Hability[Magia].Name.XPos = 390
 Habilities.Hability[Magia].Name.YPos = 226

 Habilities.DistributionPoints.XPos = 360
 Habilities.DistributionPoints.YPos = 330
 Habilities.DistributionPoints.PercentageBar.XPos = 355
 Habilities.DistributionPoints.PercentageBar.YPos = 330
 Habilities.DistributionPoints.PercentageBar.HorizontalDrawFlag = true
 Habilities.DistributionPoints.PercentageBar.Lenght = 150
 Habilities.DistributionPoints.PercentageBar.Width = 14
 Habilities.DistributionPoints.PercentageBar.Alpha = 150

 Fade.Start.RedColor = 0
 Fade.Start.GreenColor = 0
 Fade.Start.BlueColor = 0
 Fade.In.Speed = 196
 Fade.Out.OnCancel.RedColor = 0
 Fade.Out.OnCancel.GreenColor = 0
 Fade.Out.OnCancel.BlueColor = 0
 Fade.Out.OnCancel.Speed = 196
 Fade.Out.OnAccept.RedColor = 0
 Fade.Out.OnAccept.GreenColor = 0
 Fade.Out.OnAccept.BlueColor = 0
 Fade.Out.OnAccept.Speed = 196
End
```

B.9.12. Sección [GameLoading].

En esta sección se establecerá la configuración del interfaz que se activará cuando comience una nueva partida, o cuando desde la interfaz de cargar partidas se elija una para recupearla. Este interfaz únicamente mostrará una imagen de fondo y un posible texto estático indicando que se está en proceso de carga. La novedad será que podrán existir una o más imágenes (sin posibilidad de asociar animación) para mostrarlas de forma aleatoria.

Así pues, en la variable `BackImage[x].FileName` se establecerá una imagen de fondo siendo x un valor entre 0 y el número máximo de imágenes de fondo asociadas menos uno.

Además del dato anterior, se deberá de completar la información de fade. Como esta interfaz se activará después de que en el estado anterior de juego (en la interfaz anterior) se haya realizado un fade a un determinado color (un fade out) será necesario especificar la velocidad a la que se realizará un fade desde ese color hasta la visualización normal de la imagen aleatoria que se haya cargado. Ese valor se establecerá en `Fade.In.Speed` y deberá de valer más de 0 y oscilar, como recomendación, entre 128 y 192 para una transición suave. Una vez que se haya cargado un juego, se realizará un fade a un color específico que se establecerá en las variables `Fade.Out.RedColor`, `Fade.Out.GreenColor` y `Fade.Out.BlueColor` usando una velocidad que será estáblecida en la variable `Fade.Out.Speed` siguiendo todas las indicaciones anteriormente expliadas para `Fade.In.Speed`.

Ejemplo:

```
[GameLoading]
 BackImage[0].FileName = Graphics\Interfaz\AreaLoadingBackImg.tga
 BackImage[1].FileName = Graphics\Interfaz\AreaLoadingBackImg2.tga

 Fade.In.Speed = 256
 Fade.Out.RedColor = 0
 Fade.Out.GreenColor = 0
 Fade.Out.BlueColor = 0
 Fade.Out.Speed = 256
End
```

B.9.13. Sección [ConversationWindow].

En esta sección se definirá la configuración del interfaz asociado a la conversación. Este interfaz será, en cierto modo, muy parecido al utilizado para representar el interfaz de juego principal pues ocupará todo su espacio en la zona inferior de la pantalla, teniendo la imagen de fondo las mismas características, en cuanto a dimensiones, que las que posee la utilizada en el interfaz de juego principal.

Ejemplo de ventana de conversación

El primer elemento de configuración será el de la imagen de fondo. Únicamente se especificará el nombre de la imagen sobre la que se solaparán todos los demás elementos de interfaz. En la

variable `BackImage.FileName` se recogerá ese nombre. Será muy importante tener en cuenta que las dimensiones de esta imagen deberán de ser, necesariamente, de una anchura igual a la resolución horizontal en la que se esté construyendo el motor, esto es, 640, 800 o 1024 y una altura de 96 pixels. El motor colocará automáticamente a esta interfaz en la parte inferior de pantalla. Además, la imagen de fondo no deberá de tener ningún color invisible asociado pues estará tapando una zona del mapa que jamás se dibujará.

A continuación se esperará leer la configuración relativa a cómo serán mostradas las líneas con las posibles opciones de conversación. Esas opciones de conversación jugarán el papel de los fragmentos que el jugador podrá decir en un determinado momento. Todas las variables relativas, se formarán utilizando el prefijo `TextOptionsFormat.`. Así, para indicar en qué posición aparecerá la primera línea visible con la que el motor luego ordenará todas las demas, usaremos las terminaciones `XPos` e `YPos`. Para indicar el número máximo de líneas visibles se usará la terminación `MaxLines`, cuyo valor deberá de oscilar entre 1 y 6. El área que podrán ocupar las líneas a lo ancho será definido añadiendo la terminación `AreaWidth` y comenzará a contar desde la coordenada de dibujado en X que se estableció utilizando la variable `TextOptionsFormat.XPos`. Para indicar que una opción de diálogo está seleccionada se establecerá el color leido añadiendo las terminaciones `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor` mientras que en el caso de que una línea de diálogo no se halle apuntada por el cursor, el color utilizado será el que se haya leido de las variables formadas con las terminaciones `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor`. Por último, todo el área que podrán ocupar las líneas de díalogo a elegir, estará sobre un FXBack cuya configuración se leerá añadiendo las terminaciones `FXBack.RedColor`, `FXBack.GreenColor` y `FXBack.BlueColor` para establecer su color y `FXBack.Alpha` para asociar el grado de transparencia asociado al mismo.

Para poder navegar entre las opciones harán falta dos botones, uno para desplazar líneas hacia arriba y otro para desplazar líneas hacia abajo. Mediante las variables cuyo prefijo sea `Options.UpButton.` configuraremos el botón de desplazamiento hacia arriba y mediante las variables cuyo prefijo sea `Options.DownButton.` haremos lo propio con el botón de desplazamiento hacia abajo. Bastará añadir las terminaciones `AnimTemplate`, `XPos` e `YPos` para crear las variables que indiquen la plantilla de animacion y las posiciones.

El último apartado de configuración será el relativo a mostrar los retratos y nombres de los interlocutores. Cómo es lógico, este interfaz siempre se activará siendo el jugador uno de los citados interlocutores. Tanto el apartado destinado al jugador como el destinado al resto de posibles entidades con los que éste pueda hablar, poseerán la misma configuración, así que a la hora de especificar los valores se utilizarán prefijos distintos pero terminaciones iguales. Mediante el prefijo `Speakers.Player.` se crearán las variables de configuración del jugador y mediante `Speakers.Rest.` se hará lo propio con el resto de entidades. Añadiendo las terminaciones `Portrait.XPos`, `Portrait.YPos` localizaremos el retrato, con `Name.RedColor`, `Name.GreenColor` y `Name.BlueColor` estableceremos el color del nombre y con `Name.XPos` y `Name.YPos` asociaremos la posición del mismo.

Ejemplo:

```
[ConversationWindow]
 BackImage.FileName = Graphics\Interfaz\MIDialog.tga

 TextOptionsFormat.XPos = 210
 TextOptionsFormat.YPos = 394
 TextOptionsFormat.MaxLines = 5
 TextOptionsFormat.AreaWidth = 345
 TextOptionsFormat.Select.RedColor = 255
 TextOptionsFormat.Select.GreenColor = 255
 TextOptionsFormat.Select.BlueColor = 255
 TextOptionsFormat.Unselect.RedColor = 180
 TextOptionsFormat.Unselect.GreenColor = 180
 TextOptionsFormat.Unselect.BlueColor = 200
 TextOptionsFormat.FXBack.RedColor = 100
 TextOptionsFormat.FXBack.GreenColor = 0
 TextOptionsFormat.FXBack.BlueColor = 0
```

```
 TextOptionsFormat.FXBack.Alpha = 155

 Speakers.Player.Portrait.XPos = 10
 Speakers.Player.Portrait.YPos = 393
 Speakers.Player.Name.RedColor = 255
 Speakers.Player.Name.GreenColor = 255
 Speakers.Player.Name.BlueColor = 0
 Speakers.Player.Name.XPos = 12
 Speakers.Player.Name.YPos = 458
 Speakers.Rest.Portrait.XPos = 118
 Speakers.Rest.Portrait.YPos = 410
 Speakers.Rest.Name.RedColor = 255
 Speakers.Rest.Name.GreenColor = 255
 Speakers.Rest.Name.BlueColor = 0
 Speakers.Rest.Name.XPos = 120
 Speakers.Rest.Name.YPos = 394

 Options.UpButton.AnimTemplate = ATUpButton
 Options.UpButton.XPos = 575
 Options.UpButton.YPos = 398
 Options.DownButton.AnimTemplate = ATDownButton
 Options.DownButton.XPos = 575
 Options.DownButton.YPos = 435
End
```

B.9.14. Sección [EntityInteract].

En esta sección se configurará un interfaz que será usado de forma intensiva durante el juego, pues su función será la de dirigir la interacción del jugador con las entidades. Desde el se podrá ordenar si se quiere observar, hablar, usar un ítem, ejecutar una habilidad, etc. El interfaz estará, a grandes rasgos, formado por un menú flotante que aparecerá formado por una serie de botones con las opciones que el jugador podrá realizar sobre una entidad. Cuando se coloque el cursor sobre alguna de dichas opciones, además de pasarse a estado de selección, un posible valor de transparencia que se podrá asociar desaparecerá para hacer completamente distinguible el botón de opción escogido. Ademas un texto flotante ayudará a indicar qué tipo de acción representa el icono seleccionado. En el caso de que se quiera utilizar una habilidad hablaremos de un menú distinto que estará formado por botones de opciones relativos a la habilidad a utilizar.

Ejemplo del menú de interacción (el jugador interactúa con un baúl)

Así pues, los primeros valores a establecer serán en global, esto es, afectarán tanto al menú de opciones de interacción estándar como al de uso de habilidades. Uno de esos valores se referirá al grado de transparencia asociado a aquellos botones de opciones que no sean seleccionados por el cursor. El valor se asociará en la variable `DrawFormat.Options.Unselect.Alpha`. Por otro lado, se podrá establecer el color de texto flotante a utilizar mediante `FloatingText.RedColor`, `FloatingText.GreenColor` y `FloatingText.BlueColor`.

El siguiente paso será la construcción del menú de opciones de interacción. Sólo habrá que suministrar la plantilla de animación con el botón en cuestión. Dicha plantilla de animación, además de cumplir los requisitos de cualquier botón, deberá de tener un tamaño del orden de 32x32 para no constituir botones excesivamente “toscos”. La variable a utilizar será `MainMenu.Option[x].AnimTemplate` donde `x` podrá valer `Observe` para la opción

observar, `Talk` para la opción de hablar, `Get` para la opción de recoger un ítem del suelo, `Register` para registrar una entidad, `Manipulate` para manipularla, `DropItem` para soltar un ítem, `UseInventory` para usar un objeto de inventario, `UseHability` para usar una habilidad y `Exit` para salir del menú. Nótese que el bóton asociado a `UseHability` sólo habrá que establecerlo si en el archivo de reglas se definió al menos un identificador de habilidad. Si las habilidades no existen no hará falta establecer nada.

El siguiente paso será el definir el menú asociado al uso de las habilidades. Naturalmente, si no existen habilidades en las reglas de juego, se podrá pasar por alto. Para construir el menú bastará usar la variable `HabilityMenu.Option[y].AnimTemplate` donde y será el identificador de una habilidad. Una vez asociados los botones de habilidad, se establecerá el relativo al retorno de ese menú al anterior mediante la variable `HabilityMenu.Option[ReturnToMainMenu].AnimTemplate`.

Ejemplo:

```
[EntityInteract]
 DrawFormat.Options.Unselect.Alpha = 175

 FloatingText.RedColor = 255
 FloatingText.GreenColor = 255
 FloatingText.BlueColor = 255

 MainMenu.Option[Observe].AnimTemplate = ATObserve
 MainMenu.Option[Talk].AnimTemplate = ATTalk
 MainMenu.Option[Get].AnimTemplate = ATGet
 MainMenu.Option[Register].AnimTemplate = ATRegister
 MainMenu.Option[Manipulate].AnimTemplate = ATManipulate
 MainMenu.Option[DropItem].AnimTemplate = ATDrop
 MainMenu.Option[UseInventory].AnimTemplate = ATUseInventory
 MainMenu.Option[UseHability].AnimTemplate = ATUseHabilities
 MainMenu.Option[Exit].AnimTemplate = ATExit

 HabilityMenu.Option[Magia].AnimTemplate = ATHabilityMagic
 HabilityMenu.Option[ReturnToMainMenu].AnimTemplate = ATHabilityReturn
End
```

B.9.15. Sección [InventorySelector].

En esta sección se configurará el interfaz que aparecerá cuando la opción de usar un ítem de inventario sea escogida. En este caso, se mostrará una ventana con todos los ítems que posea el jugador. Si éste selecciona uno de ellos y pulsa el botón izquierdo, el personaje controlado por el jugador recibirá la orden de usar dicho ítem sobre la entidad destino con la que estaba interactuando. También existirá la opción de cerrar la ventana y olvidar la posibilidad de interactuar (esto ocurrirá irremediablemente si se eligió esta opción sin tener ningún ítem en inventario).

El primer paso será establecer la imagen de fondo y asociarla a una posición y un grado de transparencia. Para realizar esta configuración, se utilizarán las variables `BackImage.FileName` para el nombre de la imagen y `BackImage.XPos`, `BackImage.YPos` y `BackImage.Alpha` para su posición y grado de transparencia respectivamente.

El siguiente paso será definir la zona en donde se mostrarán los ítems de inventario. Esta zona se creará utilizando un concepto de matriz, es decir, indicaremos cuantas filas y columnas de selectores o ítems se mostraran. Las variables `ItemSelector.Area.Width` y `ItemSelector.Area.Height` servirán para tal fin porque la primera de ellas indicará la anchura del área destinado a los selectores de ítems (en unidades). El valor podrá oscilar entre 1 y 8. La otra variable lo que hará será indicar la altura del área y su valor oscilará entre 1 y 6. Para entenderlo, pensemos que si valieran 3 y 2 respectivamente, estariamos viendo 2 filas con 3 selectores cada una, en total hasta 6 ítems visibles en pantalla. El siguiente paso será definir la posición en la que comenzarán a verse los selectores a través de las variables `ItemSelector.XPos` e `ItemSelector.YPos`. Por último, se indicará la configuración del FXBack que se activará cuando un ítem sea seleccionado. En todos los casos, se contará con

el prefijo `ItemSelector.FXBack.` de esta forma, usando las terminaciones `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor` indicaremos el color RGB y con `Select.Alpha` el grado de transparencia. El color del texto flotante que aparecerá para mostrar el nombre del ítem seleccionado se configurará en las variables cuyo prefijo sea `ItemSelector.FloatingText.` y cuya terminación aparezca como `RedColor`, `GreenColor` y `BlueColor`.

Ejemplo de una posible interfaz

El último paso será el de configurar los botones de opciones. Las opciones posibles en este interfaz serán las relativas a cerrarlo y desplazar hacia arriba o hacia abajo los ítems que sean mostrados, esto es, el inventario del jugador. Lo primero, cerrar el interfaz, se configurará a través de las variables cuyo prefijo sea `Options.CancelButton.`, la opción de desplazar los ítems en el inventario hacia arriba se configurará con las variables cuyo prefijo sea `Options.UpButton.` y, por último, formaremos las variables de configuración de la opción de desplazamiento de ítems hacia abajo usando el prefijo `Options.DownButton.`. Según los especificado, bastará añadir las terminaciones `AnimTemplate`, `XPos` e `YPos` en cada caso para especificar la plantilla de animación y posición, respectivamente, de cada opción.

Ejemplo:

```
[InventorySelector]
 BackImage.FileName = Graphics\Interfaz\SelectItemToUseBackImg.tga
 BackImage.XPos = 30
 BackImage.YPos = 20
 BackImage.Alpha = 200

 ItemSelector.Area.Width = 1
 ItemSelector.Area.Height = 4
 ItemSelector.XPos = 39
 ItemSelector.YPos = 55
 ItemSelector.FXBack.Select.RedColor = 155
 ItemSelector.FXBack.Select.GreenColor = 0
 ItemSelector.FXBack.Select.BlueColor = 0
 ItemSelector.FXBack.Select.Alpha = 100
 ItemSelector.FloatingText.RedColor = 255
 ItemSelector.FloatingText.GreenColor = 200
 ItemSelector.FloatingText.BlueColor = 0

 Options.UpButton.AnimTemplate = ATUpButton
 Options.UpButton.XPos = 115
 Options.UpButton.YPos = 230
 Options.DownButton.AnimTemplate = ATDownButton
 Options.DownButton.XPos = 115
 Options.DownButton.YPos = 272
 Options.CancelButton.AnimTemplate = ATXButton
 Options.CancelButton.XPos = 120
 Options.CancelButton.YPos = 26
End
```

B.9.16. Sección [TradeItemsManager].

Esta sección estará destinada a configurar las ventanas que aparecerán cuando se proceda a registrar, por parte del usuario, una entidad contenedor. Una entidad contenedor podrá ser desde un objeto de escenario que sea contenedor (un baúl) hasta una criatura que haya muerto y el jugador quiera investigar en su contenido. Hablamos de ventanas porque la idea será ofrecer una con el contenido de inventario de la entidad a la que el jugador va a investigar y otra con el propio contenido del jugador. Mientras que en la ventana asociada a la entidad a la que el jugador investiga no podremos hacer otra cosa más que observar sus ítems y pasarlos a nuestro inventario, desde nuestra ventana podremos pasarlos hacia la otra, observarlos y tirarlos. Esta ventana también gestionará el cierre de la interfaz.

Ejemplo de interfaz

En primer lugar se configurará la ventana del jugador. El primer paso será establecer la imagen de fondo de la ventana sí como su posición y el grado de transparencia. Todo ello se realizará utilizando las variables cuyo prefijo sea `PlayerWindow.BackImage.` y estén completadas con las terminaciones `FileName`, `XPos`, `YPos` y `Alpha` para indicar el nombre de la imagen así como la posición y valor de transparencia respectivamente.

Una vez establecida la imagen de fondo, se configurará el área destinado a mostrar los ítems de inventario. Esta zona se creará utilizando un concepto de matriz, es decir, indicaremos cuantas filas y columnas de selectores o ítems se mostraran. Las variables se formarán utilizando los prefijos `PlayerWindow.ItemSelector.` de esta forma, utilizando las terminaciones `Area.Width` y `Area.Height` se podrá indicar la anchura del área destinado a los selectores de ítems en unidades (el valor oscilará entre 1 y 8) y la altura (su valor oscilará entre 1 y 6). Para entenderlo, pensemos que si valieran 3 y 2 respetivamente, estariamos viendo 2 filas con 3 selectores cada una, en tontal hasta 6 ítems visibles en pantalla. El siguiente paso será definir la posición en la que comenzarán a verse los selectores a través de las variables cuya terminación sea `XPos` e `YPos`. Por último, se indicará la configuración del FXBack que se activará cuando un ítem sea seleccionado. Con las terminaciones `FXBack.Select.RedColor`, `FXBack.Select.GreenColor` y `FXBack.Select.BlueColor` indicaremos el color RGB y con `FXBack.Select.Alpha` el grado de transparencia. El color del texto flotante que aparecerá para mostrar el nombre del ítem seleccionado se configurará en las terminaciones `FloatingText.RedColor`, `FloatingText.GreenColor` y `FloatingText.BlueColor`.

El último paso en la configuración de la ventana del jugador será establecer los botones de opciones dependientes de la misma. Los dos primeros serán para navegar por el contenido de su inventario y el último para especificar el botón de cancelación. En el caso del desplazamiento hacia arriba utilizaremos variables cuyo prefijo sea `PlayerWindow.Options.UpButton.`, en el caso del desplazamiento hacia abajo `PlayerWindow.Options.DownButton.` y en el caso del botón de cancelación

`PlayerWindow.Options.CancelButton.`. Como en todos los casos se necesitará especificar lo mismo, bastará añadir als terminaciones `AnimTemplate`, `XPos` e `YPos` con el fin de indicar la plantilla de animación y posición respectivamente.

El siguiente paso será el relativo a configurar la ventana del inventario asociado a la entidad sobre la que el jugador interactúe. Los pasos a seguir serán idénticos y las variables tendrán el mismo nombre salvo que en lugar de comenzar por `PlayerWindow` comenzarán por `RestWindow` para indicar jugado y resto de interlocutores respectivamente. Al ser iguales se omitirá volver a poner la misma explicación. La única novedad con respecto a esta ventana será que existirá una zona en donde se mostrará el retrato de la entidad (si es que se asoció en el archivo de perfil de entidades) con la que estamos interactuando. A través de las variables con terminación `Portrait.XPos` y `Portrait.YPos` indicaremos la posición donde mostrar la plantilla de animación con el posible retrato, mientras que usando las terminaciones `Portrait.Name.XPos`, `Portrait.Name.YPos`, `Portrait.Name.RedColor`, `Portrait.Name.GreenColor` y `Portrait.Name.BlueColor` indicaremos la posición y el color RGB con el que mostrar la posición y el nombre de la entidad con la que el jugador interactue.

Ejemplo:

```
[TradeItemsManager]
 PlayerWindow.BackImage.FileName = Graphics\Interfaz\BackImage_1.tga
 PlayerWindow.BackImage.XPos = 340
 PlayerWindow.BackImage.YPos = 50
 PlayerWindow.BackImage.Alpha = 200

 PlayerWindow.ItemSelector.Area.Width = 3
 PlayerWindow.ItemSelector.Area.Height = 3
 PlayerWindow.ItemSelector.XPos = 350
 PlayerWindow.ItemSelector.YPos = 99
 PlayerWindow.ItemSelector.FXBack.Select.RedColor = 155
 PlayerWindow.ItemSelector.FXBack.Select.GreenColor = 0
 PlayerWindow.ItemSelector.FXBack.Select.BlueColor = 0
 PlayerWindow.ItemSelector.FXBack.Select.Alpha = 100
 PlayerWindow.ItemSelector.FloatingText.RedColor = 255
 PlayerWindow.ItemSelector.FloatingText.GreenColor = 200
 PlayerWindow.ItemSelector.FloatingText.BlueColor = 0

 PlayerWindow.Options.UpButton.AnimTemplate = ATUpButton
 PlayerWindow.Options.UpButton.XPos = 550
 PlayerWindow.Options.UpButton.YPos = 215
 PlayerWindow.Options.DownButton.AnimTemplate = ATDownButton
 PlayerWindow.Options.DownButton.XPos = 550
 PlayerWindow.Options.DownButton.YPos = 257
 PlayerWindow.Options.CancelButton.AnimTemplate = ATXButton
 PlayerWindow.Options.CancelButton.XPos = 559
 PlayerWindow.Options.CancelButton.YPos = 55

 RestWindow.BackImage.FileName = Graphics\Interfaz\BackImage_2.tga
 RestWindow.BackImage.XPos = 50
 RestWindow.BackImage.YPos = 50
 RestWindow.BackImage.Alpha = 200

 RestWindow.ItemSelector.Area.Width = 2
 RestWindow.ItemSelector.Area.Height = 3
 RestWindow.ItemSelector.XPos = 62
 RestWindow.ItemSelector.YPos = 85
 RestWindow.ItemSelector.FXBack.Select.RedColor = 155
 RestWindow.ItemSelector.FXBack.Select.GreenColor = 0
 RestWindow.ItemSelector.FXBack.Select.BlueColor = 0
 RestWindow.ItemSelector.FXBack.Select.Alpha = 100
 RestWindow.ItemSelector.FloatingText.RedColor = 255
 RestWindow.ItemSelector.FloatingText.GreenColor = 200
 RestWindow.ItemSelector.FloatingText.BlueColor = 0

 RestWindow.Portrait.XPos = 219
```

```
 RestWindow.Portrait.YPos = 67
 RestWindow.Portrait.Name.XPos = 219
 RestWindow.Portrait.Name.YPos = 135
 RestWindow.Portrait.Name.RedColor = 255
 RestWindow.Portrait.Name.GreenColor = 255
 RestWindow.Portrait.Name.BlueColor =  255

 RestWindow.Options.UpButton.AnimTemplate = ATUpButton
 RestWindow.Options.UpButton.XPos = 200
 RestWindow.Options.UpButton.YPos = 200
 RestWindow.Options.DownButton.AnimTemplate = ATDownButton
 RestWindow.Options.DownButton.XPos = 200
 RestWindow.Options.DownButton.YPos = 242
End
```

B.9.17. Sección [CombatientInfo].

En esta sección se establecerá la información relativa al interfaz de combate. El interfaz de combate será meramente informativo y se activará tan solo cuando el juego esté en situación de “pelea”. Básicamente toda la información que mostrará este interfaz será el de una barra de porcentaje que controlará el tanto por cien de los puntos de acción que tiene la criatura actual mas luego un texto con el valor actual y el valor base de dichos puntos como complemento a esa barra de porcentaje.

La configuración de la barra de porcentaje se realizará a partir de las variables `ActionPoints.PercentageBar.XPos` y `ActionPoints.PercentageBar.YPos` para indicar la posición en donde mostrarla, `ActionPoints.PercentageBar.Lenght`, `ActionPoints.PercentageBar.Width` para indicar su longitud y anchura y, por último, a través de `ActionPoints.PercentageBar.Alpha` para indicar su componente de transparencia. El texto con los valores base y temporal se pondrá automáticamente sobre la barra de porcentaje y únicamente será necesario especificar aquí el color RGB con el que se querrá que aparezca en pantalla a partir de las variables `ActionPoints.Value.RedColor`, `ActionPoints.Value.GreenColor` y `ActionPoints.Value.BlueColor`.

Ejemplo:

```
[CombatientInfo]
 ActionPoints.PercentageBar.XPos = 125
 ActionPoints.PercentageBar.YPos = 10
 ActionPoints.PercentageBar.Lenght = 100
 ActionPoints.PercentageBar.Width = 15
 ActionPoints.PercentageBar.Alpha = 100
 ActionPoints.Value.RedColor = 155
 ActionPoints.Value.GreenColor = 200
 ActionPoints.Value.BlueColor = 0
End
```

B.9.18. Sección [MainGameInterfaz].

En esta sección se configurará el interfaz principal de juego. Este interfaz estará encarnado por la ventana que aparece en la parte inferior de la pantalla y sobre la que se situará la consola de juego, el retrato del jugador, los slots principales de equipamiento así como los botones relativos al control del combate. Desde esta interfaz se podrá también acceder al inventario del jugador pulsando en el retrato de éste. Durante este documento, se han mostrado multitud de imágenes de ejemplo con esta interfaz.

El panel horario, así como la activación de los iconos de síntomas activos también será algo controlado desde este interfaz.

El primer elemento de configuración será el de la imagen de fondo. Únicamente se especificará el nombre de la imagen sobre la que se solaparán todos los demás elementos de interfaz. En la variable `BackImage.FileName` se recogerá la misma. Será muy importante tener en cuenta que las dimensiones de esta imagen deberán de ser, necesariamente, de una anchura igual a

la resolución horizontal en la que se esté construyendo el motor, esto es, 640, 800 o 1024 y una altura de 96 pixels. El motor colocará automáticamente a esta interfaz en la parte inferior de pantalla. Además, la imagen de fondo no deberá de tener ningún color invisible asociado pues estará tapando una zona del mapa que jamás se dibujará.

El siguiente elemento de configuración será el relativo a indicar el color de texto flotante que se utilizará durante el juego cuando el usuario decida interactuar sobre una entidad y se muestre al lado del cursor el nombre de la misma. Para ello, se utilizarán las variables `FloatingText.RedColor`, `FloatingText.GreenColor` y `FloatingText.BlueColor`.

La consola será una parte muy importante del interfaz de juego ya que a través de ella se podrán mandar mensajes al jugador y éste, además, podrá navegar por su contenido utilizando dos botones de desplazamiento. Todas las variables relacionadas con la consola comenzarán por `Console.`. En primer lugar se configurará el aspecto visual del texto utilizando las variables formadas por las terminaciones `Text.RedColor`, `Text.GreenColor` y `Text.BlueColor` para definir el color RGB del mismo. Con las terminaciones `XPos`, `YPos` y `Width` indicaremos el lugar en donde comenzará a escribirse la primera línea de texto de la consola. El motor controlará el número máximo de lineas que serán visibles a la vez en pantalla. Para poder clarificar mejor el texto se utilizará un FXBack asociado, cuya configuración se recogerá a través de las variables con terminación `FXBack.RedColor`, `FXBack.GreenColor` y `FXBack.BlueColor` para definir el color RGB y `FXBack.Alpha` para hacer lo propio con el nivel de transparencia. De forma opcional se podrá asociar un sonido WAV a la consola que se reproducirá de forma lineal cuando ésta registre un nuevo aviso. El nombre del sonido WAV será recogido a través de `WAVBipFileName.` Recordemos que en caso de no añadir nada no se tendrá en cuenta esta posibilidad. Los últimos apartados estarán destinados a especificar los botones de opciones para navegar por el contenido de la consola. Con las variable formadas usando las terminaciones `Options.UpButton.AnimTemplate`, `Options.UpButton.XPos` y `Options.UpButton.YPos` especificaremos el botón para desplazarnos hacia arriba por el texto y con `Options.DownButton.AnimTemplate`, `Options.DownButton.XPos` y `Options.DownButton.YPos` haremos lo propio para desplazarnos hacia abajo.

El siguiente paso estará destinado a configurar los slots de equipamiento. Un slot de equipamiento estará representado por un componente al que se le podrá asociar un ítem (esto se hará automáticamente por parte del motor cuando sea equipado) pero que tendrá un FXBack que podrá alterarse en su color y transparencia dependiendo de si es o no seleccionado por el cursor. En todos los casos, utilizaremos el prefijo `Equipment.DrawSlotsFormat.` para formar las variables. Así pues, en primer lugar se definirá el formato que tendrán los dos slots de equipamiento con respecto a su FXBack utilizando las variables formadas por las terminaciones `FXBack.Select.RedColor`, `FXBack.Select.GreenColor`, `FXBack.Select.BlueColor` y `FXBack.Select.Alpha` para el caso de estar seleccionados y `FXBack.Unselect.RedColor`, `FXBack.Unselect.GreenColor`, `FXBack.Unselect.BlueColor` y `FXBack.Unselect.Alpha` para cuando no lo estén. En ambos casos, primero se establecerá el color de RGB y luego el grado de transparencia.

Una vez establecido el formato de dibujado común para los slots, se irá directamente a completar de forma particular la configuración de cada slot por separado. Usando los prefijos `Equipment.Slot[x].` podremos formar las variables con las que configurar estos slots. Asi, con las variables formadas usando las terminaciones `XPos` y `YPos` definiremos la posición del slot mientras que con `DefaultAnimTemplate` deberemos de asociar una plantilla de animación, con dimensión igual a la de los ítems (ver apartado *Conceptos sobre el formato de los datos*), que será la que se muestre cuando no exista ningún ítem equipado. Como en este caso se entenderá que se llevan las manos vacías, bastaría con mostrar unas manos (una plantilla de animación de un solo estado y un solo frame, por ejemplo). Una vez definido estos datos del slot, se indicaría la configuración para el texto con el nombre del ítem equipado. Para ello, primero se establecerá la posición en la que aparecerá tal texto a través de las variables con las terminaciones `Name.XPos` y `Name.YPos` y luego el color, usando `Name.Select.RedColor`, `Name.Select.GreenColor` y `Name.Select.BlueColor` para

el caso de selección del slot y `Name.Unselect.RedColor`, `Name.Unselect.GreenColor` y `Name.Unselect.BlueColor` cuando el slot no se halle seleccionado. En todos los casos mostrados, el valor de x deberá de ser uno de los dos identificadores obligatorios para los slots principales.

El retrato del jugador cumplirá dos funciones. Una será meramente estética y otra será la de actuar de selector para que cuando el usuario lo seleccione con el cursor y pulse el botón izquierdo, se pueda acceder al inventario del personaje que controla. Al ser un slot su configuración pasará por asociar las coordenadas en las que se dibujará y los valores del FXBack para una selección y una deselección del mismo. Todas las variables se formarán utilizando el prefijo `PlayerPortrait.`. Asi, en lo relativo a posición tendremos las terminaciones `XPos` y `YPos` mientras que para los valores de FXBack asociados a la selección se usarán `FXBack.Select.RedColor`, `FXBack.Select.GreenColor` y `FXBack.Select.BlueColor` para establecer el color y `FXBack.Select.Alpha` para localizar el valor de transparencia. En el caso de no selección, las variables de FXBack serán aquellas formadas por las terminaciones `FXBack.Unselect.RedColor`, `FXBack.Unselect.GreenColor` y `FXBack.Unselect.BlueColor` para indicar el color RGB y `FXBack.Unselect.Alpha` para hacer lo propio con el nivel de transparencia.

En el interfaz de juego principal el diseñador también podrá mostrar, de manera opcional, dos atributos extendidos del jugador más la propia salud. Cuando se habla de mostrar información, nos estaremos refiriendo a mostrar el nombre, el valor numérico del atributo y una barra de porcentaje asociada. Todas las variables aquí comentadas comenzarán por el prefijo `Attribute[x].`. Para configurar la posición y nombre donde se mostrará el atributo, se utilizarán las variables cuyas terminaciónes sean `Name.XPos`, `Name.YPos`, `Name.RedColor`, `Name.GreenColor` y `Name.BlueColor` respectivamente. La configuración relativa a la posición y color del texto con el valor numérico asociado al atributo se establecerá las terminaciones mediante `Value.XPos`, `Value.YPos`, `Value.RedColor`, `Value.GreenColor` y `Value.BlueColor` respectivamente. Por último, la configuración de la barra de porcentaje se hará utilizando `PercentageBar.XPos` y `PercentageBar.YPos` para establecer la posición de dibujado de la misma, `PercentageBar.HorizontalDrawFlag` para indicar si se desea que se dibuje de forma horizontal (`true`) o vertical (`false`), `PercentageBar.Width` y `PercentageBar.Lenght` para establecer al anchura y altura de la misma en pixels (teniendo en cuenta que los valores deberán de oscilar entre 32 y 256) y `PercentageBar.Alpha` para asociar un grado de transparencia. En todos los casos mostrados, el valor de x podrá ser un identificador de un atributo extendido en el archivo de reglas o bien, la palabra `Health` para indicar que se quiere asociar esa configuración al atributo de salud.

Las opciones accesibles durante combate serán relativas a la posibilidad de huir de éste (flee) y a la de pasar de turno cuando el que lo posea sea el jugador y no quiera terminar de gastar todos sus puntos de acción. Todas las variables de configuración se establecerán utilizando el prefijo `Combat.Options.` así, para establecer los botones asociados a dichas acciones se usarán las variables con las terminación `NextTurnButton.AnimTemplate`, `NextTurnButton.XPos` y `NextTurnButton.YPos` en el caso del botón de pasar de turno y `FleeButton.XPos`, `FleeButton.YPos` y `FleeButton.AnimTemplate` en el caso del botón de huir de combate.

Si en el archivo de reglas se definió uno o más síntomas posibles, el menú principal ofrecerá la posibilidad de mostrar iconos inidicativos de que el jugador posee un determinado síntoma. La idea es que estos iconos aparezcan en alguna zona de la pantalla en la que no molesten y su misión sea meramente de carácter informativo, para comunicar al jugador que se ha activado un síntoma o que ya le ha desaparecido, sin tener que acudir a la pantalla de inventario. Mediante las variables `Symptoms.Icon[x].AnimTemplate`, `Symptoms.Icon[x].XPos` y `Symptoms.Icon[x].YPos` podremos asociar una plantilla de animación y una posición para dibujar al síntoma x, siendo x un identificador de síntoma definido en el archivo de reglas. En este caso, las plantillas de animación deberán de tener al menos un estado de animación. De forma global, se podrá asociar un valor de transparencia a dichas plantillas para que cuando

sean dibujadas en pantalla y el personaje controlado por el jugador pase debajo de ellas, no resulten molestas. Esto se logrará a través de la variable `Symptoms.Alpha`.

Ejemplo de colocación de síntomas activos por encima de la interfaz de juego principal

El último apartado de configuración del interfaz, estará destinado a trabajar con el panel de control horario. Este panel tendrá como objetivo mostrar visualmente los cambios horarios y permitir que pueda ser seleccionado y pulsado. La finalidad de que pueda ser tratado como un botón es que al pulsarse se generará un evento en virtud del cual se podrá activar desde script un cuadro selector de texto en donde el diseñador pueda ofrecer al jugador dormir. Las tres primeras variables a establecer tendrán un carácter global pues afectarán a todos los botones asociados a las distintas horas.

Ejemplo del panel horario (situado en la esquina superior derecha del área de juego)

Mediante la variable `TimeController.Unselect.Alpha` se indicará el valor de transparencia asociado cuando el panel de turno no esté seleccionado. Con las variables `TimeController.XPos` y `TimeController.YPos` indicaremos la posición en donde todos los paneles serán dibujados. A partir de aquí sólo hay que asociar distintos botones a las distintas horas para ello, se utilizará la variable `TimeController.TimeButton[x].AnimTemplate` la cual, esperará recibir el identificador de una plantilla de animación con, al menos, dos estados, pues no será más que un botón. El valor de x oscilará entre 0 y 23 refiriéndose en cada caso a una hora del día.

Ejemplo:

```
[MainGameInterfaz]
 BackImage.FileName = Graphics\Interfaz\MIBackground.tga

 FloatingText.RedColor = 255
 FloatingText.GreenColor = 200
 FloatingText.BlueColor = 0

 Console.Text.RedColor = 255
 Console.Text.GreenColor = 200
 Console.Text.BlueColor = 0
 Console.XPos = 8
 Console.YPos = 395
 Console.Width = 176
 Console.WAVBipFileName = ..\WAV\bip.WAV
 Console.Options.UpButton.AnimTemplate = ATUpButton
 Console.Options.UpButton.XPos = 192
 Console.Options.UpButton.YPos = 397
 Console.Options.DownButton.AnimTemplate = ATDownButton
 Console.Options.DownButton.XPos = 192
 Console.Options.DownButton.YPos = 437
 Console.FXBack.RedColor = 200
 Console.FXBack.GreenColor = 155
 Console.FXBack.BlueColor = 0
 Console.FXBack.Alpha = 75
```

```
Equipment.DrawSlotsFormat.FXBack.Select.RedColor = 155
Equipment.DrawSlotsFormat.FXBack.Select.GreenColor = 0
Equipment.DrawSlotsFormat.FXBack.Select.BlueColor = 0
Equipment.DrawSlotsFormat.FXBack.Select.Alpha = 100
Equipment.DrawSlotsFormat.FXBack.Unselect.RedColor = 0
Equipment.DrawSlotsFormat.FXBack.Unselect.GreenColor = 0
Equipment.DrawSlotsFormat.FXBack.Unselect.BlueColor = 100
Equipment.DrawSlotsFormat.FXBack.Unselect.Alpha = 100

Equipment.Slot[ManoIzquierda].XPos = 246
Equipment.Slot[ManoIzquierda].YPos = 402
Equipment.Slot[ManoIzquierda].DefaultAnimTemplate = ATLeftHandIcon
Equipment.Slot[ManoIzquierda].Name.XPos = 236
Equipment.Slot[ManoIzquierda].Name.YPos = 387
Equipment.Slot[ManoIzquierda].Name.Select.RedColor = 200
Equipment.Slot[ManoIzquierda].Name.Select.GreenColor = 255
Equipment.Slot[ManoIzquierda].Name.Select.BlueColor = 0
Equipment.Slot[ManoIzquierda].Name.Unselect.RedColor = 200
Equipment.Slot[ManoIzquierda].Name.Unselect.GreenColor = 200
Equipment.Slot[ManoIzquierda].Name.Unselect.BlueColor = 200

Equipment.Slot[ManoDerecha].XPos = 410
Equipment.Slot[ManoDerecha].YPos = 402
Equipment.Slot[ManoDerecha].DefaultAnimTemplate = ATRightHandIcon
Equipment.Slot[ManoDerecha].Name.XPos = 400
Equipment.Slot[ManoDerecha].Name.YPos = 387
Equipment.Slot[ManoDerecha].Name.Select.RedColor = 200
Equipment.Slot[ManoDerecha].Name.Select.GreenColor = 255
Equipment.Slot[ManoDerecha].Name.Select.BlueColor = 0
Equipment.Slot[ManoDerecha].Name.Unselect.RedColor = 200
Equipment.Slot[ManoDerecha].Name.Unselect.GreenColor = 200
Equipment.Slot[ManoDerecha].Name.Unselect.BlueColor = 200

PlayerPortrait.XPos = 325
PlayerPortrait.YPos = 395
PlayerPortrait.FXBack.Select.RedColor = 155
PlayerPortrait.FXBack.Select.GreenColor = 0
PlayerPortrait.FXBack.Select.BlueColor = 0
PlayerPortrait.FXBack.Select.Alpha = 100
PlayerPortrait.FXBack.Unselect.RedColor = 0
PlayerPortrait.FXBack.Unselect.GreenColor = 0
PlayerPortrait.FXBack.Unselect.BlueColor = 0
PlayerPortrait.FXBack.Unselect.Alpha = 100

Combat.Options.NextTurnButton.XPos = 589
Combat.Options.NextTurnButton.YPos = 395
Combat.Options.NextTurnButton.AnimTemplate = ATCombatSigTurn
Combat.Options.FleeButton.XPos = 589
Combat.Options.FleeButton.YPos = 432
Combat.Options.FleeButton.AnimTemplate = ATCombatFlee

Symptoms.Alpha = 175
Symptoms.Icon[Envenenado].AnimTemplate = ATSymptomEnvenenado
Symptoms.Icon[Envenenado].XPos = 20
Symptoms.Icon[Envenenado].YPos = 350
Symptoms.Icon[Cansado].AnimTemplate = ATSymptomCansado
Symptoms.Icon[Cansado].XPos = 55
Symptoms.Icon[Cansado].YPos = 350
Symptoms.Icon[Maldito].AnimTemplate = ATSymptomMaldito
Symptoms.Icon[Maldito].XPos = 90
Symptoms.Icon[Maldito].YPos = 350
Symptoms.Icon[Hambriento].AnimTemplate = ATSymptomHambriento
Symptoms.Icon[Hambriento].XPos = 125
Symptoms.Icon[Hambriento].YPos = 350

Attributes.Attribute[Health].Name.XPos = 485
Attributes.Attribute[Health].Name.YPos = 450
```

```
 Attributes.Attribute[Health].Name.RedColor = 200
 Attributes.Attribute[Health].Name.GreenColor = 255
 Attributes.Attribute[Health].Name.BlueColor = 0
 Attributes.Attribute[Health].Value.XPos = 485
 Attributes.Attribute[Health].Value.YPos = 460
 Attributes.Attribute[Health].Value.RedColor = 200
 Attributes.Attribute[Health].Value.GreenColor = 200
 Attributes.Attribute[Health].Value.BlueColor = 0
 Attributes.Attribute[Health].PercentageBar.XPos = 483
 Attributes.Attribute[Health].PercentageBar.YPos = 400
 Attributes.Attribute[Health].PercentageBar.HorizontalDrawFlag = false
 Attributes.Attribute[Health].PercentageBar.Lenght = 75
 Attributes.Attribute[Health].PercentageBar.Width = 10
 Attributes.Attribute[Health].PercentageBar.Alpha = 200

 TimeController.Unselect.Alpha = 200
 TimeController.XPos = 507
 TimeController.YPos = 5
 TimeController.TimeButton[0].AnimTemplate = ATTimeController_0h
 TimeController.TimeButton[1].AnimTemplate = ATTimeController_1h
 TimeController.TimeButton[2].AnimTemplate = ATTimeController_2h
 TimeController.TimeButton[3].AnimTemplate = ATTimeController_3h
 TimeController.TimeButton[4].AnimTemplate = ATTimeController_4h
 TimeController.TimeButton[5].AnimTemplate = ATTimeController_5h
 TimeController.TimeButton[6].AnimTemplate = ATTimeController_6h
 TimeController.TimeButton[7].AnimTemplate = ATTimeController_7h
 TimeController.TimeButton[8].AnimTemplate = ATTimeController_8h
 TimeController.TimeButton[9].AnimTemplate = ATTimeController_9h
 TimeController.TimeButton[10].AnimTemplate = ATTimeController_10h
 TimeController.TimeButton[11].AnimTemplate = ATTimeController_11h
 TimeController.TimeButton[12].AnimTemplate = ATTimeController_12h
 TimeController.TimeButton[13].AnimTemplate = ATTimeController_13h
 TimeController.TimeButton[14].AnimTemplate = ATTimeController_14h
 TimeController.TimeButton[15].AnimTemplate = ATTimeController_15h
 TimeController.TimeButton[16].AnimTemplate = ATTimeController_16h
 TimeController.TimeButton[17].AnimTemplate = ATTimeController_17h
 TimeController.TimeButton[18].AnimTemplate = ATTimeController_18h
 TimeController.TimeButton[19].AnimTemplate = ATTimeController_19h
 TimeController.TimeButton[20].AnimTemplate = ATTimeController_20h
 TimeController.TimeButton[21].AnimTemplate = ATTimeController_21h
 TimeController.TimeButton[22].AnimTemplate = ATTimeController_22h
 TimeController.TimeButton[23].AnimTemplate = ATTimeController_23h
End
```

B.9.19. Sección [GameMenu].

En esta sección se configurará el menú de juego. El menú de juego aparecerá siempre que se pulse la tecla ESC desde el escenario de juego. Acto seguido, el universo se pausará y no se reactivará hasta que se vuelva a regresar al mismo. Desde este interfaz o menú se podrá guardar una partida, cargar otra previamente salvada, salir del juego para regresar al menú principal o bien retornar al juego quitando la pausa introducida.

En primer lugar se esperará leer la configuración de la imagen de fondo. Con la variable `BackImage.FileName` se indicará el nombre de la imagen de fondo y mediante, `BackImage.XPos` y `BackImage.YPos` la posición en donde comenzará a ser dibujada. Por último, con `BackImage.Alpha` se establecerá el grado de transparencia, pudiendo asociar como máximo un valor de 255 que es solidez absoluta.

El siguiente paso será leer la configuración de los botones de opciones. Como siempre, los botones de opciones esperarán recibir la información relativa al identificador de la plantilla de animación que los representa y las coordenada de dibujado de los mismos. El botón de guardar partidas se configurará a través de las variables cuyo prefijo sea `Options.SaveButton.`, el de cargar partidas usando `Options.LoadButton.` , el de retornar al juego quitando la pausa usando `Options.ReturnButton.`, mientras que el de abandonar el juego actual para regresar al menú principal `Options.ExitButton.`. Todas las variables se construirán

utilizando las terminaciones `AnimTemplate`, `XPos` y `YPos` para especificar las plantillas de animación y las coordenadas de posición.

Ejemplo de la interfaz de juego

La última información que se esperará leer será la relativa a la información de fade que se aplicará cuando se de al botón se salir de juego. En este caso, se realizará un fade hacia un color RGB especificado en las variables `Fade.Out.RedColor`, `Fade.Out.GreenColor` y `Fade.Out.BlueColor` usando una velocidad especificada en la variable `Fade.Out.Speed` cuyo valor deberá de ser mayor que 0 y como recomendación, será bueno que oscile entre 128 y 192 para conseguir una transición suave a ese color.

Ejemplo:

```
[GameMenu]
 BackImage.FileName = Graphics\Interfaz\GameMenuBack.TGA
 BackImage.XPos = 192
 BackImage.YPos = 75
 BackImage.Alpha = 200

 Options.SaveButton.AnimTemplate = ATGameMenuSaveOption
 Options.SaveButton.XPos = 308
 Options.SaveButton.YPos = 108
 Options.LoadButton.AnimTemplate = ATGameMenuLoadOption
 Options.LoadButton.XPos = 308
 Options.LoadButton.YPos = 148
 Options.ReturnButton.AnimTemplate = ATGameMenuResumeOption
 Options.ReturnButton.XPos = 308
 Options.ReturnButton.YPos = 228
 Options.ExitButton.AnimTemplate = ATGameMenuExitGameOption
 Options.ExitButton.XPos = 308
 Options.ExitButton.YPos = 268

 Fade.Out.RedColor = 0
 Fade.Out.GreenColor = 0
 Fade.Out.BlueColor = 0
 Fade.Out.Speed = 196
End
```

B.9.20. Sección [PlayerProfileWindow].

En esta sección se establecerá la configuración de la interfaz relacionada con el inventario del jugador. Su complejidad será una de las mayores debido a que en ella se mostrará, además del inventario del jugador, otra información y opciones complementarias e imprescindibles como el trabajo con los slots de equipamiento, información de valores asociados a atributos, habilidades o síntomas.

En primer lugar se deberá de establecer la configuración de la imagen de fondo mediante las variables cuyo prefijo sea `BackImage.`. De esta forma, especificaremos el nombre de la imagen utilizando la terminación `FileName` y asociaremos animaciones estáticas usaremos las terminaciones `AnimTemplate`, `XPos` e `YPos`, siendo x un valor oscilante entre 0 y 255.

Ejemplo de una pantalla de inventario de jugador

El color del texto flotante que aparecerá al señalar un ítem se configurará a través de las variables cuyo prefijo sea `FloatingText.` y su terminación `RedColor`, `GreenColor` y `BlueColor`.

La consola será una parte muy importante del interfaz de juego ya que a través de ella se podrán mandar mensajes al jugador y éste, además, podrá navegar por su contenido utilizando dos botones de desplazamiento. Todas las variables relacionadas con la consola comenzarán por `Console.`. En primer lugar se configurará el aspecto visual del texto utilizando las variables formadas por las terminaciones `Text.RedColor`, `Text.GreenColor` y `Text.BlueColor` para definir el color RGB del mismo. Con las terminaciones `XPos`, `YPos` y `.Width` indicaremos el lugar en donde comenzará a escribirse la primera línea de texto de la consola. El motor controlará el número máximo de lineas que serán visibles a la vez en pantalla. Para poder clarificar mejor el texto se utilizará un FXBack asociado, cuya configuración se recogerá a través de las variables con terminación `FXBack.RedColor`, `FXBack.GreenColor` y `FXBack.BlueColor` para definir el color RGB y `FXBack.Alpha` para hacer lo propio con el nivel de transparencia. De forma opcional se podrá asociar un sonido WAV a la consola que se reproducirá de forma lineal cuando ésta registre un nuevo aviso. El nombre del sonido WAV será recogido a través de `WAVBipFileName`. Recordemos que en caso de no añadir nada no se tendrá en cuenta esta posibilidad. Los últimos apartados estarán destinados a especificar los botones de opciones para navegar por el contenido de la consola. Con las variable formadas usando las terminaciones `Options.UpButton.AnimTemplate`, `Options.UpButton.XPos` y `Options.UpButton.YPos` especificaremos el botón para desplazarnos hacia arriba por el texto y con `Options.DownButton.AnimTemplate`, `Options.DownButton.XPos` y `Options.DownButton.YPos` haremos lo propio para desplazarnos hacia abajo.

El siguiente paso será configurar el área en donde los ítems serán mostrados y seleccionados. Esta zona se creará utilizando un concepto de matriz, es decir, indicaremos cuantas filas y columnas de selectores o ítems se mostraran. Las variables se construirán sobre el prefijo `Inventory.ItemSelector`. Así, las terminaciones `Area.Width` y `Area.Height` servirán para indicar la dimensión de la matriz de selección, la primera de ellas indicará la anchura del área destinado a los selectores de ítems en unidades (el valor oscilará entre 1 y 8) y la segunda lo que hará será indicar la altura del área (su valor oscilará entre 1 y 6). Para entenderlo, pensemos que si valieran 3 y 2 respetivamente, estariamos viendo 2 filas con 3 selectores cada una, en tontal hasta 6 ítems visibles en pantalla. El siguiente paso será definir la posición en la que comenzarán a verse los selectores a través de las variables con terminaciones `XPos` e `YPos`. Por último, se indicará la configuración del FXBack que se activará cuando un ítem sea seleccionado. Con las terminaciones `FXBack.Select.RedColor`, `FXBack.Select.GreenColor` y `FXBack.Select.BlueColor` indicaremos el color RGB y con `FXBack.Select.Alpha` el

grado de transparencia. El color del texto flotante que aparecerá para mostrar el nombre del ítem seleccionado se configurará en las variables cuya terminación sea `FloatingText.RedColor`, `FloatingText.GreenColor` y `FloatingText.BlueColor`. El último paso en la configuración será el de establecer los botones de navegación por el área destinada al inventario. Para configurar el botón de desplazamiento hacia arriba se utilizarán el prefijo `Inventory.Options.UpButton`, mientras que el botón de desplazar hacia abajo se configurará utilando `Inventory.Options.DownButton`. Así, utilizando las terminaciones `AnimTemplate`, `XPos` e `YPos` tendremos las variables que permitan configurar los botones de opción anteriores.

A continuación se establecerán los iconos de síntomas. Se esperará que estos iconos de síntomas sean los mismos que se asociaron en el interfaz de juego principal. La particularidad residirá en que, de forma global, tendrán asociados un valor de transparencia que se establecerá en aquellos iconos que el jugador no posea. Si el jugador tiene un síntoma, el icono se verá de forma sólida y cuando se pase el cursor sobre el mismo saldrá en el texto flotante el nombre del síntoma. Este método se utilizará para distinguir entre síntomas activos e inactivos. Todas las variables comenzarán por el prefijo `Symptoms.`. El valor de transparencia se establecerá usando la variable `NoPresent.Alpha` mientras que para indicar la plantilla de animación con el icono de síntoma y su posición, utilizaremos las variables cuya terminación sea `Symptom[x].AnimTemplate`, `Symptom[x].XPos` y `Symptom[x].YPos` siendo `x` un identificador de síntoma definido en el archivo de reglas.

El siguiente paso será asociar los iconos de habilidades y su posición. Se esperará que se establezcan los mismos iconos definidos en la interfaz de creación del personaje y el menú de interacción (el interfaz siempre establecerá el estado 0 en las plantillas de animación). La interfaz sólo mostrará los iconos de habilidad que posee el jugador al contrario de lo que hacía con los síntomas (mostrar todos los iconos usando un nivel de transparencia para los que no se poseían). Las variables a utilizar para establecer la configuración de los iconos de habilidades utilizarán el prefijo `Habilities.` y las terminaciones `Hability[x].AnimTemplate`, `Hability[Magia].XPos` y `Hability[Magia].YPos` para establecer plantilla de animación y posición respectivamente.

El retrato del jugador volverá a hacer acto de presencia en este interfaz pues la forma de usar un ítem de inventario sobre el jugador será tomarlo de éste y llevarlo hasta el retrato. Para configurar el retrato del jugador, deberemos de tener en cuenta que este aparecerá asociado a un selector, por lo que existirá un FXBack asociado. Este FXBack tendrá configuración cuando el retrato no sea seleccionado, esto es, no se pose el cursor del ratón sobre el mismo, y cuando sí sea seleccionado. Todas las variables relativas al retrato del jugador, utilizarán un prefijo `PlayerPortrait.`. Así, para establecer la posición de dibujado del selector se utilizarán las terminaciones `XPos` e `YPos` mientras que para configurar el FXBack cuando haya selección, se rellenarán las variables cuya terminación sea `FXBack.Select.RedColor`, `FXBack.Select.GreenColor` y `FXBack.Select.BlueColor` para el color y `FXBack.Select.Alpha` para el nivel de transparencia. En el caso de la configuración del FXBack para cuando no esté seleccionado el retrato, las variables a utilizar serán aquellas cuya terminación sea `FXBack.Unselect.RedColor`, `FXBack.Unselect.GreenColor` y `FXBack.Unselect.BlueColor` para el color y `FXBack.Unselect.Alpha` para el nivel de transparencia.

La configuración de los slots de equipamiento será muy importante ya que desde este interfaz se podrán equipar los ítems. Todos los slots de equipamiento se representarán a través de selectores y tendrán asociados un nombre. Cuando se coloque el cursor sobre el selector, el color del texto asociado al nombre podrá ser distinto así como el propio color del FXBack asociado por defecto a todo selector. Para formar las variables se utilizará el prefijo `EquipmentSlots.DrawFormat.Names.`. Así, usando las variables con terminación `Select.RedColor`, `Select.GreenColor` y `Select.BlueColor` estableceremos el color a usar para los nombres de los slots de equipamiento cuando éstos sean seleccionados y con `Unselect.RedColor`, `Unselect.GreenColor` y `Unselect.BlueColor` haremos lo propio para cuando los nombres no sean seleccionados. Estas variables sólo se tendrán en cuenta sí la variable formada conla terminación `Names.UseFlag` posee un valor `true`. Esto es así

porque servirá para indicar si se desea que aparezcan los nombres de los slots de equipamiento o no. La configuración del FXBack propio de los selectores en cuanto a su color y valor de transparencia se configurará, en el caso de la selección, a través de las variables formadas utilizando el prefijo `EquipmentSlots.DrawFormat.FXBack.Select.` para la selección y `EquipmentSlots.DrawFormat.FXBack.Unselect.` para la no selección. Las terminaciones se completarán utilizando `RedColor`, `GreenColor`, `BlueColor` y `Alpha` mientras que en el caso de la no selección, se utilizará `RedColor`, `GreenColor`, `BlueColor` y `Alpha`.

A la hora de configurar de forma particular cada uno de los selectores, se utilizarán las variables que comiencen por el prefijo `EquipmentSlots.Slot[x].`. Así, se utilizarán las terminaciones `XPos` e `YPos` para indicar la posición de los mismos. En el caso de utilizar los nombres, también se esperará configurar la posición en la que aparecerán éstos a través de las variables con terminación `Name.XPos` y `Name.YPos`. En todos los casos, x será el identificador de un slot de equipamiento definido en el archivo de reglas. Recordemos, que toda criatura al menos poseerá dos slots de equipamiento que representarán a sus manos.

Despues de los slots de equipamiento, sólo quedará como paso complejo configurar el apartado destinado a los atributos. Cuando hablamos de atributos, hablamos de mostrar el nombre de salud, tipo, clase, nivel, experiencia y atributos extendidos que posea el jugador, sus valores numéricos y las barras de porcentaje asociadas (esto último sólo para el caso de la salud y los atributos extendidos). En primer lugar se establecerá una configuración global, en cuanto color y, en el caso de las barras de porcentaje, formato, para mostrar los nombres, valores numéricos y barras de porcentaje. Todos los valores aquí comentados se establecerán utilizando variables cuyo prefijo sea `Attributes.DrawFormat.`. Así, con las terminaciones `Names.RedColor`, `Names.GreenColor` y `Names.BlueColor` indicaremos el color de los nombres de los atributos, mientras que con `Values.RedColor`, `Values.GreenColor` y `Values.BlueColor` haremos lo propio en lo relativo al texto que muestre los valores numéricos. Para las barras de porcentaje usaremos las variables cuyo prefijo sean `Attributes.DrawFormat.PercentageBars.`. Así, usaremos la terminación `HorizontalDrawFlag` con el fin de indicar si éstas serán dibujadas en horizontal (`true`) o vertical (`false`) y `Lenght` y `Width` para establecer su longitud y anchura, (siempre con valores oscilantes entre 32 y 256). El porcentaje de transparencia de las barras se asociará con la variable formada añadiendo la terminación `Alpha`.

Con el formato de dibujado para los atributos resuelto, sólo restará situar las posiciones. La posición del texto con el nombre y género del jugador se establecerá mediante variables cuyo prefijo será siempre `Attributes.`. De esta forma, usando las terminaciones `Player.Name.XPos`, `Player.Name.YPos`, `GenreTypeClass.Name.XPos` y `GenreTypeClass.Name.YPos` respectivamente. El texto identificativo del nivel y su valor con las terminaciones `Level.Name.XPos`, `Level.Name.YPos` y `Level.Value.XPos`, `Level.Value.YPos` respectivamente. Para hacer lo mismo con la experiencia se usarán las variables con terminación `Experience.Name.XPos`, `Experience.Name.YPos`, `Experience.Value.XPos`, `Experience.Value.YPos` respectivamente. Los puntos de combate se localizarán con las terminaciones `CombatPoints.Name.XPos` y `CombatPoints.Name.YPos` para el texto identificativo y con `CombatPoints.Value.XPos` y `CombatPoints.Value.YPos` para el valor. La salud deberá de especificarse tanto en texto identificativo a través de las terminaciones `Health.Name.XPos` y `Health.Name.YPos` como con valor numérico y de porcentaje usando `Health.Value.XPos`, `Health.Value.YPos` y `Health.PercentageBar.XPos`, `Health.PercentageBar.YPos` respectivamente. Los atributos extendidos, al igual que la salud, se asociarán para nombre, valor numérico y barra de porcentaje. Para ellos, se utilizarán las variables cuyo prefijo sea `Attributes.ExtendedAttribute[x].`. De esta forma, las terminaciones `Name.XPos` y `Name.YPos` en el caso del nombre, `Value.XPos` y `Value.YPos` en el caso del valor numérico y `PercentageBar.XPos` y `PercentageBar.YPos` en el caso de la barra de porcentaje.

El último apartado de este interfaz será el de situar el botón de salida y retorno a juego. Se hará a través de las variables cuyo prefijo sea `Options.ExitButton.`, con lo que las terminaciones `AnimTemplate`, `XPos` e `YPos` determinarán la plantilla de animación y la posición respectivamente.

Ejemplo:

```
[PlayerProfileWindow]
 BackImage.FileName = Graphics\Interfaz\ImageBack.tga

 FloatingText.RedColor = 255
 FloatingText.GreenColor = 200
 FloatingText.BlueColor = 0

 Console.Text.RedColor = 255
 Console.Text.GreenColor = 200
 Console.Text.BlueColor = 0
 Console.XPos = 87
 Console.YPos = 146
 Console.Width = 144
 Console.NumLines = 12
 Console.FXBack.RedColor = 0
 Console.FXBack.GreenColor = 0
 Console.FXBack.BlueColor = 255
 Console.FXBack.Alpha = 55
 Console.Options.UpButton.AnimTemplate = ATScrollUpButton
 Console.Options.UpButton.XPos = 163
 Console.Options.UpButton.YPos = 110
 Console.Options.DownButton.AnimTemplate = ATScrollDownButton
 Console.Options.DownButton.XPos = 199
 Console.Options.DownButton.YPos = 110

 Inventory.ItemSelector.Area.Width = 1
 Inventory.ItemSelector.Area.Height = 5
 Inventory.ItemSelector.XPos = 11
 Inventory.ItemSelector.YPos = 18
 Inventory.ItemSelector.FXBack.Select.RedColor = 155
 Inventory.ItemSelector.FXBack.Select.GreenColor = 0
 Inventory.ItemSelector.FXBack.Select.BlueColor = 0
 Inventory.ItemSelector.FXBack.Select.Alpha = 100
 Inventory.ItemSelector.FXBack.Unselect.RedColor = 155
 Inventory.ItemSelector.FXBack.Unselect.GreenColor = 155
 Inventory.ItemSelector.FXBack.Unselect.BlueColor = 155
 Inventory.ItemSelector.FXBack.Unselect.Alpha = 100
 Inventory.ItemSelector.FloatingText.RedColor = 255
 Inventory.ItemSelector.FloatingText.GreenColor = 200
 Inventory.ItemSelector.FloatingText.BlueColor = 0
 Inventory.Options.UpButton.AnimTemplate = ATUpButton
 Inventory.Options.UpButton.XPos = 83
 Inventory.Options.UpButton.YPos = 21
 Inventory.Options.DownButton.AnimTemplate = ATDownButton
 Inventory.Options.DownButton.XPos = 83
 Inventory.Options.DownButton.YPos = 64

 Symptoms.NoPresent.Alpha = 100
 Symptoms.Symptom[Envenenado].AnimTemplate = ATSymptomEnvenenado
 Symptoms.Symptom[Envenenado].XPos = 20
 Symptoms.Symptom[Envenenado].YPos = 350
 Symptoms.Symptom[Cansado].AnimTemplate = ATSymptomCansado
 Symptoms.Symptom[Cansado].XPos = 55
 Symptoms.Symptom[Cansado].YPos = 350
 Symptoms.Symptom[Maldito].AnimTemplate = ATSymptomMaldito
 Symptoms.Symptom[Maldito].XPos = 90
 Symptoms.Symptom[Maldito].YPos = 350
 Symptoms.Symptom[Hambriento].AnimTemplate = ATSymptomHambriento
 Symptoms.Symptom[Hambriento].XPos = 125
 Symptoms.Symptom[Hambriento].YPos = 350
```

```
Habilities.Hability[Magia].AnimTemplate = ATHabilityMagic
Habilities.Hability[Magia].XPos = 500
Habilities.Hability[Magia].YPos = 436

PlayerPortrait.XPos = 389
PlayerPortrait.YPos = 48
PlayerPortrait.FXBack.Select.RedColor = 155
PlayerPortrait.FXBack.Select.GreenColor = 0
PlayerPortrait.FXBack.Select.BlueColor = 0
PlayerPortrait.FXBack.Select.Alpha = 100
PlayerPortrait.FXBack.Unselect.RedColor = 0
PlayerPortrait.FXBack.Unselect.GreenColor = 0
PlayerPortrait.FXBack.Unselect.BlueColor = 0
PlayerPortrait.FXBack.Unselect.Alpha = 100

EquipmentSlots.DrawFormat.Names.UseFlag = true
EquipmentSlots.DrawFormat.Names.Select.RedColor = 200
EquipmentSlots.DrawFormat.Names.Select.GreenColor = 255
EquipmentSlots.DrawFormat.Names.Select.BlueColor = 0
EquipmentSlots.DrawFormat.Names.Unselect.RedColor = 155
EquipmentSlots.DrawFormat.Names.Unselect.GreenColor = 155
EquipmentSlots.DrawFormat.Names.Unselect.BlueColor = 155
EquipmentSlots.DrawFormat.FXBack.Select.RedColor = 155
EquipmentSlots.DrawFormat.FXBack.Select.GreenColor = 0
EquipmentSlots.DrawFormat.FXBack.Select.BlueColor = 0
EquipmentSlots.DrawFormat.FXBack.Select.Alpha = 155
EquipmentSlots.DrawFormat.FXBack.Unselect.RedColor = 155
EquipmentSlots.DrawFormat.FXBack.Unselect.GreenColor = 155
EquipmentSlots.DrawFormat.FXBack.Unselect.BlueColor = 155
EquipmentSlots.DrawFormat.FXBack.Unselect.Alpha = 155

EquipmentSlots.Slot[ManoIzquierda].XPos = 310
EquipmentSlots.Slot[ManoIzquierda].YPos = 17
EquipmentSlots.Slot[ManoIzquierda].Name.XPos = 310
EquipmentSlots.Slot[ManoIzquierda].Name.YPos = 2
EquipmentSlots.Slot[ManoDerecha].XPos = 466
EquipmentSlots.Slot[ManoDerecha].YPos = 17
EquipmentSlots.Slot[ManoDerecha].Name.XPos = 466
EquipmentSlots.Slot[ManoDerecha].Name.YPos = 2
EquipmentSlots.Slot[Cabeza].XPos = 310
EquipmentSlots.Slot[Cabeza].YPos = 84
EquipmentSlots.Slot[Cabeza].Name.XPos = 310
EquipmentSlots.Slot[Cabeza].Name.YPos = 150
EquipmentSlots.Slot[Vestido].XPos = 466
EquipmentSlots.Slot[Vestido].YPos = 84
EquipmentSlots.Slot[Vestido].Name.XPos = 466
EquipmentSlots.Slot[Vestido].Name.YPos = 150

Attributes.DrawFormat.Names.RedColor = 225
Attributes.DrawFormat.Names.GreenColor = 225
Attributes.DrawFormat.Names.BlueColor = 0
Attributes.DrawFormat.Values.RedColor = 200
Attributes.DrawFormat.Values.GreenColor = 200
Attributes.DrawFormat.Values.BlueColor = 200
Attributes.DrawFormat.PercentageBars.HorizontalDrawFlag = true
Attributes.DrawFormat.PercentageBars.Lenght = 100
Attributes.DrawFormat.PercentageBars.Width = 10
Attributes.DrawFormat.PercentageBars.Alpha = 175

Attributes.Player.Name.XPos = 280
Attributes.Player.Name.YPos = 180
Attributes.GenreTypeClass.Name.XPos = 280
Attributes.GenreTypeClass.Name.YPos = 195
Attributes.Level.Name.XPos = 280
Attributes.Level.Name.YPos = 210
Attributes.Level.Value.XPos = 400
Attributes.Level.Value.YPos = 213
Attributes.Experience.Name.XPos = 280
```

```
 Attributes.Experience.Name.YPos = 225
 Attributes.Experience.Value.XPos = 400
 Attributes.Experience.Value.YPos = 228
 Attributes.CombatPoints.Name.XPos = 280
 Attributes.CombatPoints.Name.YPos = 240
 Attributes.CombatPoints.Value.XPos = 400
 Attributes.CombatPoints.Value.YPos = 243
 Attributes.Health.Name.XPos = 280
 Attributes.Health.Name.YPos = 255
 Attributes.Health.Value.XPos = 400
 Attributes.Health.Value.YPos = 258
 Attributes.Health.PercentageBar.XPos = 397
 Attributes.Health.PercentageBar.YPos = 257
 Attributes.ExtendedAttribute[Destreza].Name.XPos = 280
 Attributes.ExtendedAttribute[Destreza].Name.YPos = 270
 Attributes.ExtendedAttribute[Destreza].Value.XPos = 400
 Attributes.ExtendedAttribute[Destreza].Value.YPos = 273
 Attributes.ExtendedAttribute[Destreza].PercentageBar.XPos = 397
 Attributes.ExtendedAttribute[Destreza].PercentageBar.YPos = 272
 Attributes.ExtendedAttribute[Fuerza].Name.XPos = 280
 Attributes.ExtendedAttribute[Fuerza].Name.YPos = 285
 Attributes.ExtendedAttribute[Fuerza].Value.XPos = 400
 Attributes.ExtendedAttribute[Fuerza].Value.YPos = 288
 Attributes.ExtendedAttribute[Fuerza].PercentageBar.XPos = 397
 Attributes.ExtendedAttribute[Fuerza].PercentageBar.YPos = 287
 Attributes.ExtendedAttribute[Suerte].Name.XPos = 280
 Attributes.ExtendedAttribute[Suerte].Name.YPos = 300
 Attributes.ExtendedAttribute[Suerte].Value.XPos = 400
 Attributes.ExtendedAttribute[Suerte].Value.YPos = 303
 Attributes.ExtendedAttribute[Suerte].PercentageBar.XPos = 397
 Attributes.ExtendedAttribute[Suerte].PercentageBar.YPos = 302

 Options.ExitButton.AnimTemplate = ATPlayerProfileExitButton
 Options.ExitButton.XPos = 602
 Options.ExitButton.YPos = 2
End
```

B.10. El archivo de definiciones GFX.

El archivo de definiciones GFX servirá para establecer parámetros de efectos especiales gráficos y para crear las plantillas de GFX. Mientras que lo segundo será necesario sólo si se va a utilizar alguna en el juego, lo primero resultará obligatorio para poder definir una serie de valores internos con los que deberá de trabajar el motor.

B.10.1. Sección [FXDefs].

En esta sección se establecerán los efectos especiales con los que funcionará el motor. El primero de ellos será el referido a la luz ambiente. Recordemos que cuando se carga un área en el que la luz ambiente tiene valor 0, el transcurso horario se deberá de reflejar en el escenario. A través de la variable `AmbientDark.Hour[x].Dark` se podrá establecer un valor de oscuridad que tendrá la hora x, siendo x un valor entre 0 y 23 (indicando este valor el rango horario al que está asociado el valor luminoso). Si queremos oscuridad total bastará con asociar un valor de 255, mientras que acercándonos a 0 estaremos asignando valores de claridad.

Desde esta sección también se podrá controlar la intensidad con la que las sombras serán dibujadas en pantalla. Las sombras se asocian en los archivos de perfil destinados a entidades como paredes, objetos de escenario, npcs o jugador y serán siempre gráficos (normalmente siluetas de color negro). Como es lógico, a medida que amanece el valor de sombra asociado será menor con respecto a cuando estamos al medio día. Lo mismo ocurrirá cuando avanzamos hacia la noche. El objeto de la variable `Shadows.Hour[x].Alpha` será el de asociar el valor de transparencia (alpha) al gráfico de sombra deacuerdo a un valor horario entre 0 y 23, que es justamente lo que representa la x. Los valores numéricos que se podrán asignar podrán ir desde 0 a 255 siendo el valor de 0 transparencia absoluta (no se ve el gráfico de sombra) y 255 solidez absoluta (se ve tal y como es el gráfico de sombra).

El valor alpha de transparencia para aquellas criaturas que tienen el efecto de transparencia activado se establecerá usando la variable `Criatures.TransparentMode.Alpha`.

Para asociar el valor de transparencia en las paredes que ocultan al jugador y se hacen invisibles de modo automático, usaremos `Walls.AutoTransparenteMode.Alpha`. Nótese que en caso de colocar un valor de 255 haremos que las paredes queden completamente sólidas. Lo normal será usar un valor de 125.

Ejemplo:

```
[FXDefs]
 AmbientDark.Hour[0].Dark = 255
 AmbientDark.Hour[1].Dark = 255
 AmbientDark.Hour[2].Dark = 255
 AmbientDark.Hour[3].Dark = 255
 AmbientDark.Hour[4].Dark = 255
 AmbientDark.Hour[5].Dark = 200
 AmbientDark.Hour[6].Dark = 155
 AmbientDark.Hour[7].Dark = 100
 AmbientDark.Hour[8].Dark = 55
 AmbientDark.Hour[9].Dark = 0
 AmbientDark.Hour[10].Dark = 0
 AmbientDark.Hour[11].Dark = 0
 AmbientDark.Hour[12].Dark = 0
 AmbientDark.Hour[13].Dark = 0
 AmbientDark.Hour[14].Dark = 0
 AmbientDark.Hour[15].Dark = 0
 AmbientDark.Hour[16].Dark = 0
 AmbientDark.Hour[17].Dark = 0
 AmbientDark.Hour[18].Dark = 0
 AmbientDark.Hour[19].Dark = 55
 AmbientDark.Hour[20].Dark = 100
 AmbientDark.Hour[21].Dark = 155
 AmbientDark.Hour[22].Dark = 200
 AmbientDark.Hour[23].Dark = 255
```

```
 Shadows.Hour[0].Alpha = 0
 Shadows.Hour[1].Alpha = 0
 Shadows.Hour[2].Alpha = 0
 Shadows.Hour[3].Alpha = 0
 Shadows.Hour[4].Alpha = 0
 Shadows.Hour[5].Alpha = 0
 Shadows.Hour[6].Alpha = 10
 Shadows.Hour[7].Alpha = 40
 Shadows.Hour[8].Alpha = 70
 Shadows.Hour[9].Alpha = 100
 Shadows.Hour[10].Alpha = 130
 Shadows.Hour[11].Alpha = 160
 Shadows.Hour[12].Alpha = 175
 Shadows.Hour[13].Alpha = 175
 Shadows.Hour[14].Alpha = 175
 Shadows.Hour[15].Alpha = 160
 Shadows.Hour[16].Alpha = 130
 Shadows.Hour[17].Alpha = 100
 Shadows.Hour[18].Alpha = 70
 Shadows.Hour[19].Alpha = 40
 Shadows.Hour[20].Alpha = 10
 Shadows.Hour[21].Alpha = 0
 Shadows.Hour[22].Alpha = 0
 Shadows.Hour[23].Alpha = 0

 Criatures.TransparentMode.Alpha = 125
 Walls.AutoTransparenteMode.Alpha = 125
End
```

B.10.2. Secciones de perfiles de GFX.

Estas secciones tendrán el mismo formato que cualquier otra sección de perfil, esto es, `[Profile<Nombre>]` siendo `Nombre` la denominación que realmente da el diseñador al perfil que está definiendo. Los perfiles de GFX estarán determinados por una plantilla de animación que se asocia a una entidad y a la que se le aplican diversos efectos especiales.

La primera de las variables a establecer es `AnimTemplate` y en ella se indicará el nombre del perfil de la plantilla de animación que, como es lógico, deberá de figurar en el archivo de plantillas de animación.

A continuación se deberá de especificar el modo de dibujado de la plantilla en relación a la entidad a la que se va asociar, para ello existirán 4 variables lógicas, pudiendo valer sólo una de ellos `true`. Las variables comenzarán siempre por el prefijo `DrawMode.` siendo sus terminaciones `OverEntityTopFlag` para indicar que se quiere dibujar la plantilla de animación por encima de la entidad, `AtEntityTopFlag` para indicar que se quiere dibujar la animación a la misma altura que la entidad, `AtEntityBaseFlag` para indicar que se quiere dibujar a la altura de la base de la entidad a la que se asocie y `UnderEntityBaseFlag` para indicar que la animación se deberá de dibujar por debajo de la entidad. Por defecto todas valdrán `false`. En el caso de que no se establezca ninguna, el motor hará como si se hubiera leído el valor `true` sobre la variable `DrawMode.AtEntityTopFlag`. Como el orden de lectura de las variables es el expuesto, si tanto la variable `DrawMode.OverEntityTopFlag` como `DrawMode.UnderEntityBaseFlag` estuvieran a `true`, el motor escogerá la variable `DrawMode.OverEntityTopFlag`.

Después de indicar el modo de dibujado, se podrá establecer un efecto especial asociado a los distintos vértices que forman el quad en donde irán asociados los gráficos de los frames, mediante la modificación del color RGB y el valor de transparencia asociado. Para lograrlo se contará con las variables cuyo prefijo sea `FX.Vertex[x].` de tal forma que las terminaciones `Alpha`, `RedColor`, `GreenColor` y `BlueColor` siendo x un valor entre 0 y 3 para indicar sobre que vértice, de los 4 posibles, se quiere actuar. En todos los casos los valores numéricos posibles oscilarán entre 0 y 255. En la variable cuya terminación sea `Alpha` estableceremos el

valor de transparencia que será total cuando valga 0 y no existirá cuando sea 255. En las variables restantes estableceremos el color RGB asociado a dicho vértice. Cuando el valor de éstas sea de 255 simultáneamente, estaremos indicando que no modificamos su valor por defecto. Nótese que en todos los casos el valor por defecto será de 255. Para más información sobre los valores alpha, colores y vértices consultar la sección *Conceptos sobre el formato de los datos*.

El último de los valores posibles estará relacionado al modo de terminación. Un efecto de GFX podrá finalizar por tiempo o bien podrá finalizar de forma manual, es decir, mediante la ejecución de una orden que lo desvincule de la entidad a la que fue asociado. La variable `EndMode.ByTime` servirá para establecer el valor temporal que existirá antes de que el efecto de GFX sea quitado. Por defecto valdrá 0 y a no ser que se le asocie un valor mayor o igual a 1 no será efectivo con respecto al método manual. El tiempo siempre será medido en segundos.

Ejemplo:

```
[GFXProfile<MagiaRayo>]
 AnimTemplate = ATGFXMagiaRayo
 DrawMode.AtEntityTopFlag = true
 FX.Vertex[0].Alpha = 0
 FX.Vertex[0].RedColor = 0
 FX.Vertex[0].GreenColor = 255
 FX.Vertex[0].BlueColor = 0
 FX.Vertex[3].Alpha = 255
 FX.Vertex[3].RedColor = 255
 FX.Vertex[3].GreenColor = 0
 FX.Vertex[3].BlueColor = 0
 EndMode.ByTime = 15
End
```

B.11. El archivo de plantillas de presentación.

Las plantillas de presentación tienen el objetivo de poder crear secuencias de imágenes con animaciones y sonido para formar con ellas presentaciones sencillas pero efectivas a la hora de mostrar una determinada narración, una imagen, etc. El concepto general será el de diapositiva (slide) ya que cada estado de la presentación será realmente una diapositiva. Al igual que otros archivos de plantillas, las secciones tendrán siempre el formato `[Profile<Nombre>]` siendo `Nombre` un identificador que el usuario asociará a dicha plantilla. Al menos deberá de existir un perfil de presentación que será usado para mostrar los créditos.

Todo perfil de presentación deberá de comenzar con el establecimiento de unos valores base de configuración. Las variables a utilizar serán `Configuration.NumberOfSlides`, que se utilizará para indicar el número de diapositivas existentes en la presentación (siempre deberá de valer al menos 1), `Configuration.ESCKeyActiveFlag`, que servirá para habilitar la tecla `ESC` y permitir al usuario salir de la presentación actual y retornar al estado del motor anterior y `Configuration.MaxSlideVisualizationTime`, que servirá para activar el pase automático de diapositivas y cuyo valor por defecto será de 0, lo que indicará que para pasar de una diapositiva a la siguiente, se deberá de pulsar al botón derecho del ratón. Si su valor fuera mayor o igual a 1, estaríamos indicando los segundos en los que cada diapositiva será visualizada antes de pasar a la siguiente.

Una vez definidos los valores de configuración, se tendrán que establecer los parámetros relativos al fade inicial que existirá en la pantalla. Por fade inicial entenderemos el color con el que se mostrará la pantalla y el valor de velocidad con la que se pasará de ese color a la imagen que constituirá la primera diapositiva. Así, en las variables con prefijo `Start.Fade.Out.` y terminación `RedColor`, `GreenColor` y `BlueColor` se establecerá ese color RGB y usando la variable `Start.Fade.In.Speed` el valor de la velocidad a la que se hará la transición de ese color a la imagen de la primera diapositiva. Es recomendable que esta velocidad esté en torno a 128 o 192 que viene a indicar el número de pasos en un segundo que se harán hacia la imagen real.

Cuando se hayan establecido los valores de configuración y fade, se pasará a definir cada una de las diapositivas. Todas seguirán el mismo patrón con la salvedad de que será siempre obligatorio que en la primera diapositiva (slide 0) se asocie una imagen de fondo. Luego en las demás diapositivas, si no se asocia una imagen de fondo siempre se utilizará la imagen que estaba anterior. En la variable `Slide[x].BackImage.FileName` se establecerá el nombre de la imagen de fondo. Para poder asociar animaciones estáticas a esa imagen de fondo, se deberán de utilizar las variables `Slide[x].Anim[y].XPos, Slide[x].Anim[y].YPos` que indicarán la posición de la animación en pantalla y `Slide[x].Anim[y].AnimTemplate` que servirá para especificar el identificador de la plantilla de animación a utilizar. El valor x representa a la diapositiva actual mientras que y representa a la animación actual. Al contrario de lo que ocurrirá con las imágenes de fondo, las animaciones no se guardarán de diapositiva a diapositiva, por lo que se deberán de volver a especificar.

A continuación se esperará recibir la posible información musical en torno a un fichero MIDI. Decimos posible información ya que no será obligatorio el que exista un fichero MIDI sonando de fondo. En caso de querer establecerlo, se deberán de asociar dos valores. Uno de esos valores irá destinado a indicar el nombre del fichero MIDI a través de la variable `Slide[x].MIDIMusic.FileName` y el otro a indicar la forma de reproducción del mismo a través de la variable `Slide[x].MIDIMusic.CyclicPlayFlag`. Si a esta variable se le asocia `true`, el fichero MIDI se reproducirá de forma cíclica (una vez que se alcance el final volverá a comenzar) y si se asocia `false`, cuando termine de sonar finalizará. Por defecto su valor será de `false`. Los ficheros MIDI sonarán siempre aunque cambiemos de diapositiva . Esto último siempre y cuando no se hayan terminado de reproducir o bien se hayan sustituido por otros. Si lo que queremos es simplemente detener un sonido MIDI que viene de otras diapositivas anteriores, bastará con usar la variable `Slide[x].MIDIMusic.StopFlag`. Si asociamos un valor `true`, cesará cualquier sonido MIDI de diapositiva anterior que esté sonando pero no tendremos posibilidad de asociar sonido MIDI en la diapositiva actual. Por defecto valdrá `false`.

También será posible asociar sonidos digitalizados, esto es, sonidos WAV a la presentación actual. Por un lado se podrá asociar un sonido WAV denominado lineal porque su funcionamiento será el de reproducirse hasta alcanzar el final y luego cesar como sonido. Únicamente se deberá de indicar el nombre de este fichero WAV a través de la variable `Slide[x].WAVSound.Linear.FileName`. No será obligatorio poner ningún sonido WAV y en caso de pasar de diapositiva y estar el sonido aún reproduciéndose, éste cesará. Por otro lado, se podrá asociar un sonido cíclico WAV. La principal diferencia con respecto a los lineales es que estos sonidos cíclicos funcionarán, en cierta forma, como los archivos MIDI. En primer lugar permanecerán aunque se cambie de diapositiva y en segundo lugar, se podrá utilizar una variable para indicar que cese su reproducción. Para indicar el nombre del sonido WAV cíclico a reproducir se utilizará la variable `Slide[x].WAVSound.Cyclic.FileName`. Si ya hubiera un sonido WAV cíclico reproduciéndose, éste será sustituido por el nuevo. Si, por el contrario, queremos cesar en esta diapositiva cualquier sonido WAV cíclico existente, usaremos la variable `Slide[x].WAVSound.Cyclic.StopFlag`. Estableciendo un valor de `true` en ésta, cesaremos cualquier reproducción existente pero no podremos asociar un fichero en la diapositiva actual. Por defecto su valor será de `false`.

El siguiente paso será la posibilidad de escribir texto en pantalla. El texto podrá existir en un fichero de texto o bien estar indicado en este mismo archivo de perfiles. Independientemente de este factor, para todo texto que se escriba será necesario especificar una posición en donde escribirlo, un área que deberá de ocupar en cuanto a anchura y una justificación del texto en dicho área. La posición se establecerá mediante las variables `Slide[x].Text[z].XPos`, `Slide[x].Text[z].YPos` para las posiciones en pantalla con respecto al a coordenada en x e y respectivamente. Con el identificador z indicaremos que su valor podrá llegar a ser de 255, con lo que será posible escribir hasta 256 textos en pantalla. A continuación se deberá de establecer el área, a lo ancho, que ocupará dicho texto. El valor mínimo será de 64 y el máximo de 1024, este valor se especificará usando la variable `Slide[x].Text[z].AreaWidth`. El siguiente paso será el de establecer la justificación del texto en pantalla, esto es, si aparece hacia la izquierda, centrado o hacia la derecha. Bastará usar `Slide[x].Text[z].Justify.CenterFlag` si se desea justificación central, `Slide[x].Text[z].Justify.RightFlag` si se desea justificación a la derecha o bien `Slide[x].Text[z].Justify.LeftFlag` si lo que se quiere es justificar a la izquierda. El orden de lectura y prioridad será el mismo que el usado para mostrar estas variables y en todos los casos por defecto existirá el valor `false` asociado. En caso de que el valor en todas estas variables de justificación sea `false`, la justificación a la izquierda será la que se tome por defecto.

El último y más importante paso en lo que respecta al texto será indicar qué es lo que se quiere escribir. Si queremos asociar una cadena de caracteres en el mismo archivo de perfiles, bastará con asociar a `Slide[x].Text[z].Text` dicha cadena, teniendo en cuenta que deberá de caber, dicho texto, en la línea en la que se lo asociamos a la variable. Si por el contrario lo que queremos es que el texto se lea de un archivo (debido a su longitud o a un determinado método de organización a ese respecto) usaremos la variable `Slide[x].Text[z].TextFileName`. Naturalmente, el fichero suministrado, además de existir y estar bien escrito en cuanto a su ruta, deberá de ser de texto.

Sólo existirán dos formas de realizar la transición desde la diapositiva actual a la siguiente que será mediante un fade o bien mediante la carga automática de la siguiente diapositiva. Dependiendo de cómo esté diseñada la presentación, podrá interesar un sistema u otro. Si queremos usar una transición basada en un fade deberemos de especificar el color RGB de dicho fade mediante las variables `Fade.Out.RedColor`, `Fade.Out.GreenColor` y `Fade.Out.BlueColor` y la velocidad a la que se realizará mediante `Fade.Out.Speed`. La velocidad deberá siempre ser mayor de 0 y como valor aconsejado pensaremos en una velocidad de 128 o 192. En caso de no especificar el color RGB para el fade, la velocidad no se leerá pues se entenderá que la transición será sin utilizar ese efecto especial.

Siempre que en la diapositiva anterior se haya establecido un fade (y esto ocurrirá siempre con la diapositiva primera ya que al comienzo siempre existirá un fade hacia un color especificado)

se deberá de especificar la velocidad a la que dicho fade se difuminará para llegar a mostrar los datos propios de la diapositiva actual. Para especificar esto se utilizará la variable `Fade.In.Speed` cuyo valor cumplirá las mismas nombras que las comentadas para la variable `Fade.Out.Speed` un poco más atrás.

Ejemplo:

```
[Profile<Creditos>]
 NumberOfSlides = 3
 ESCKeyActiveFlag = true
 MaxSlideVisualizationTime = 14;
 Start.Fade.Out.RedColor = 0
 Start.Fade.Out.GreenColor = 0
 Start.Fade.Out.BlueColor = 0
 Start.Fade.In.Speed = 128

 Slide[0].BackImage.FileName = Graphics\Presentations\slide0.tga
 Slide[0].Text[0].RedColor = 187
 Slide[0].Text[0].GreenColor = 91
 Slide[0].Text[0].BlueColor = 193
 Slide[0].Text[0].AreaWidth = 640
 Slide[0].Text[0].XPos = 50
 Slide[0].Text[0].YPos = 150
 Slide[0].Text[0].Text = http://usuarios.tripod.es/crisolengine/
 Slide[0].Anim[0].XPos = 300
 Slide[0].Anim[0].YPos = 300
 Slide[0].Anim[0].AnimTemplate = ATSLIDE_LOGOUNIOVI
 Slide[0].Anim[1].XPos = 5
 Slide[0].Anim[1].YPos = 390
 Slide[0].Anim[1].Alpha = 200
 Slide[0].Anim[1].AnimTemplate = ATSLIDE_FIRE
 Slide[0].Anim[2].XPos = 550
 Slide[0].Anim[2].YPos = 390
 Slide[0].Anim[2].Alpha = 200
 Slide[0].Anim[2].AnimTemplate = ATSLIDE_FIRE_REVERSE
 Slide[0].Text[1].RedColor = 187
 Slide[0].Text[1].GreenColor = 91
 Slide[0].Text[1].BlueColor = 193
 Slide[0].Text[1].AreaWidth = 300
 Slide[0].Text[1].XPos = 260
 Slide[0].Text[1].YPos = 370
 Slide[0].Text[1].Text = E.U.I.T.I.O. <www15.uniovi.es>
 Slide[0].Text[2].RedColor = 180
 Slide[0].Text[2].GreenColor = 180
 Slide[0].Text[2].BlueColor = 225
 Slide[0].Text[2].AreaWidth = 640
 Slide[0].Text[2].XPos = 0
 Slide[0].Text[2].YPos = 200
 Slide[0].Text[2].Justify.CenterFlag = true
 Slide[0].Text[2].TextFileName = creditsreadme.txt
 Slide[0].Text[3].RedColor = 255
 Slide[0].Text[3].GreenColor = 255
 Slide[0].Text[3].BlueColor = 255
 Slide[0].Text[3].AreaWidth = 640
 Slide[0].Text[3].XPos = 5
 Slide[0].Text[3].YPos = 465
 Slide[0].Text[3].Text = CRISOLEngine
 Slide[0].MIDIMusic.FileName = Midi\credits.mid
 Slide[0].Fade.Out.RedColor = 0
 Slide[0].Fade.Out.GreenColor = 0
 Slide[0].Fade.Out.BlueColor = 0
 Slide[0].Fade.Out.Speed = 128
 Slide[0].Fade.In.Speed = 128

 Slide[1].BackImage.FileName = Graphics\Presentations\slide1.tga
 Slide[1].Anim[0].XPos = 5
 Slide[1].Anim[0].YPos = 390
 Slide[1].Anim[0].Alpha = 200
```

```
 Slide[1].Anim[0].AnimTemplate = ATSLIDE_FIRE
 Slide[1].Anim[1].XPos = 550
 Slide[1].Anim[1].YPos = 390
 Slide[1].Anim[1].Alpha = 200
 Slide[1].Anim[1].AnimTemplate = ATSLIDE_FIRE_REVERSE
 Slide[1].Text[0].RedColor = 255
 Slide[1].Text[0].GreenColor = 255
 Slide[1].Text[0].BlueColor = 255
 Slide[1].Text[0].AreaWidth = 640
 Slide[1].Text[0].XPos = 5
 Slide[1].Text[0].YPos = 465
 Slide[1].Text[0].Text = CRISOLEngine
 Slide[1].Fade.In.Speed = 128
 Slide[1].Fade.Out.RedColor = 0
 Slide[1].Fade.Out.GreenColor = 0
 Slide[1].Fade.Out.BlueColor = 0
 Slide[1].Fade.Out.Speed = 128

 Slide[2].BackImage.FileName = Graphics\Presentations\slide2.tga
 Slide[2].Anim[0].XPos = 5
 Slide[2].Anim[0].YPos = 390
 Slide[2].Anim[0].Alpha = 200
 Slide[2].Anim[0].AnimTemplate = ATSLIDE_FIRE
 Slide[2].Anim[1].XPos = 550
 Slide[2].Anim[1].YPos = 390
 Slide[2].Anim[1].Alpha = 200
 Slide[2].Anim[1].AnimTemplate = ATSLIDE_FIRE_REVERSE
 Slide[2].Text[0].RedColor = 255
 Slide[2].Text[0].GreenColor = 255
 Slide[2].Text[0].BlueColor = 255
 Slide[2].Text[0].AreaWidth = 640
 Slide[2].Text[0].XPos = 5
 Slide[2].Text[0].YPos = 465
 Slide[2].Text[0].Text = CRISOLEngine
 Slide[2].Fade.In.Speed = 128
 Slide[2].Fade.Out.RedColor = 0
 Slide[2].Fade.Out.GreenColor = 0
 Slide[2].Fade.Out.BlueColor = 0
 Slide[2].Fade.Out.Speed = 128
End
```

B.12. El archivo de texto estático.

El archivo de texto estático tendrá por misión definir el texto que el motor deberá de mostrar en pantalla en determinadas ocasiones. De no existir este archivo, el texto debería de ir con el código fuente, haciendo completamente imposible su traducción a otros idiomas (a no ser que se cambiará en el código y se volviera a compilar todo). El texto será obligatorio y por ser texto se podrán incluir espacios, acentos y el carácter 'ñ'.

A continuación se comentarán las diferentes secciones.

B.12.1. Sección [CriatureTypes].

Se establecerá el nombre asociado a los identificadores `Type` (de tipo) asociados a las criaturas y que fueron definidos en el archivo de reglas. El formato de las variables será `Type[x]` donde `x` será un valor entre 0 y el número máximo de identificadores que se definieron menos uno (si se definieron un total de 10 identificadores, x podrá valer a lo sumo 9).

Ejemplo:

```
[CriatureTypes]
 Type[0] = Humano
 Type[1] = Hobbit
End
```

B.12.2. Sección [CriatureClasses].

Se establecerá el nombre asociado a los identificadores `Class` (de clase) asociados a las criaturas y que fueron definidos en el archivo de reglas. El formato de las variables será `Class[x]` donde `x` será un valor entre 0 y el número máximo de identificadores que se definieron menos uno (si se definieron un total de 10 identificadores, x podrá valer a lo sumo 9).

Ejemplo:

```
[CriatureClasses]
 Class[0] = Paladín
 Class[1] = Guerrero
 Class[2] = Mago
 Class[3] = Ladrón
 Class[4] = Explorador
End
```

B.12.3. Sección [CriatureGenres].

Se establecerá el nombre asociado a los identificadores `Genre` (de género) asociados a las criaturas y que fueron definidos en el archivo de reglas. El formato de las variables será `Genre[x]` donde x será un valor entre 0 y 1, ya que el número máximo de géneros permitidos es de 2.

Ejemplo:

```
[CriatureGenres]
 Genre[0] = Varón
 Genre[1] = Hembra
End
```

B.12.4. Sección [CriatureHabilities].

Se establecerá el nombre asociado a los identificadores `Hability` (de habilidades) asociados a las criaturas y que fueron definidos en el archivo de reglas. El formato de las variables será `Hability[x]` donde `x` será un valor entre 0 y el número máximo de identificadores que se definieron menos uno (si se definieron un total de 5 identificadores, x podrá valer a lo sumo 4).

Ejemplo:

```
[CriatureHabilities]
 Hability[0] = Magia
```

```
End
```

B.12.5. Sección [CriatureSymptoms].

Se establecerá el nombre asociado a los identificadores `Symptom` (de síntomas) asociados a las criaturas y que fueron definidos en el archivo de reglas. El formato de las variables será `Symptom[x]` donde x será un valor entre 0 y el número máximo de identificadores que se definieron menos uno (si se definieron un total de 5 identificadores, x podrá valer a lo sumo 4).

Ejemplo:

```
[CriatureSymptoms]
 Symptom[0] = Envenenado
 Symptom[1] = Cansado
 Symptom[2] = Maldito
 Symptom[3] = Hambriento
End
```

B.12.6. Sección [CriatureEquipmentSlots].

Se establecerá el nombre asociado a los identificadores `EquipmentSlot` (slots de equipamiento) asociados a las criaturas y que fueron definidos en el archivo de reglas. El formato de las variables será `EquipmentSlot[x]` donde x será un valor entre 0 y el número máximo de identificadores que se definieron menos uno (si se definieron un total de 5 identificadores, x podrá valer a lo sumo 4).

B.12.7. Sección [CriatureAttributes].

En esta sección se vendrá a especificar todos los datos pertenecientes a los atributos. Así, a las variables `Level`, `Experience`, `Health` y `CombatPoints` se les asociará el texto asociado al concepto de nivel, experiencia, salud y puntos de combate respectivamente. Por último, también habrá que especificar el texto asociado a los atributos extendidos. Para ello se utilizará el formato `ExtendedAttribute[x]` donde x será un valor entre 0 y el número máximo de identificadores que se definieron menos uno (si se definieron un total de 5 identificadores, x podrá valer a lo sumo 4).

Ejemplo:

```
[CriatureAttributes]
 Level = Nivel
 Experience = Experiencia
 Health = Salud
 CombatPoints = Puntos de combate
 ExtendedAttribute[0] = Destreza
 ExtendedAttribute[1] = Fuerza
 ExtendedAttribute[2] = Suerte
End
```

B.12.8. Sección [Interfaces].

En esta sección se especificará el texto estático que pueda estar vinculado a los distintos interfaces.

El interfaz de guardar partidas necesitará texto para indicar que se introduzca el nombre de la partida a guardar. La variable `SaverSavedFileManager.SavedFileDesc.InfoText` servirá para establecer dicho texto.

El interfaz de menú principal (el que aparece nada más arrancar el motor) podrá tener asociado un texto informativo para indicar la versión o cualquier otro tipo de reseña. Para lograrlo, se le asociará texto a la variable `MainMenu.VersionInfo`.

El interfaz de interacción con entidades, esto es, el que está formado por un menú flotante que surge cuando se desea interactuar sobre una entidad del escenario de juego, tendrá asociado

texto estático para ayudar al usuario a distinguir las acciones que representarán los diferentes iconos. Mediante la variable `EntityInteract.MainMenu.Option[x]` se establecerá el texto asociado a las opciones del menú principal, esto es, el que lo forman acciones que no se corresponden con habilidades. El valor de la x podrá ser `Observe`, para la opción de observar, `Talk`, para la opción de hablar, `Get`, para la opción de coger, `Register`, para la opción de registrar, `Manipulate`, para la opción de manipular, `DropItem`, para la opción de tirar, `UseInventory`, para la opción de tirar, `UseHability`, para la opción de usar habilidad y `Exit`, para la opción de abandonar el menú.

En lo referido al texto asociado al menú de habilidades, se deberán de utilizar los identificadores de habilidades que se definieron en el archivo de reglas. La variable `HabilityMenu.Option[x]` sevirá para asociar el texto estático a las distintas opciones del menú. El valor de x siempre corresponderá al identificador de una habilidad definida en el archivo de reglas. Para asociar el texto de la opción de retorno se utilizará la variable `EntityInteract.HabilityMenu.Option[ReturnToMainMenu].`

El último interfaz que poseerá texto estático asociado será el de la creación del jugador. Usará las variables `PlayerCreation.Name`, `PlayerCreation.Genre`, `PlayerCreation.Type` y `PlayerCreation.Class` para indicar el texto asociado a la palabra nombre, género, tipo y clase de criatura respectivamente.

Ejemplo:

```
[Interfaces]
 SaverSavedFileManager.SavedFileDesc.InfoText = Introduce descripción
 MainMenu.VersionInfo = CrisolEngine v:1.0 / Fernando Rodríguez
 EntityInteract.MainMenu.Option[Observe] = Observar
 EntityInteract.MainMenu.Option[Talk] = Hablar
 EntityInteract.MainMenu.Option[Get] = Coger
 EntityInteract.MainMenu.Option[Register] = Registrar
 EntityInteract.MainMenu.Option[Manipulate] = Manipular
 EntityInteract.MainMenu.Option[DropItem] = Tirar ítem
 EntityInteract.MainMenu.Option[UseInventory] = Usar inventario
 EntityInteract.MainMenu.Option[UseHability] = Usar habilidad
 EntityInteract.MainMenu.Option[Exit] = Salir
 EntityInteract.HabilityMenu.Option[Magia] = Usar magia
 EntityInteract.HabilityMenu.Option[ReturnToMainMenu] = Retornar
 PlayerCreation.Name = Nombre
 PlayerCreation.Genre = Sexo
 PlayerCreation.Type = Tipo
 PlayerCreation.Class = Clase
End
```

B.12.9. Sección [Messages].

Está sección vendrá a recoger el texto asociado a los diferentes mensajes que se podrán surgir en el motor. La variable `InsuficientActionPoints` servirá para asociar el texto que se deberá de emitir cuando se quiera realizar, por parte del jugador, una acción en combate y éste no tenga suficientes puntos para llevarla a acabo. La variable `AreaLoading` recogerá el texto mostrado cuando se esté cargando un área bien porque se comienza una partida nueva o bien porque se está recuperando un juego previamente guardado. La variable `NoSaveGamePermission` servirá para indicar que no se puede guardar la partida y `ExitGameConfirm` deberá de tener asociado el texto a mostrar en el cuadro de diálogo que aparecerá para confirmar un abandono de una partida. El uso de `CantMoveTo` estará referenido a indicar que el jugador no podrá moverse a una determinada posición.

Ejemplo:

```
[Messages]
 InsuficientActionPoints = No hay puntos de acción suficientes
 AreaLoading = Procesando...
 NoSaveGamePermission = No se puede guardar la partida
 ExitGameConfirm = ¿Estás seguro de querer salir del juego?
 CantMoveTo = No es posible alcanzar la posición...
End
```

B.13. El archivo de scripts globales.

Este archivo será muy sencillo y únicamente estará destinado a indicar los scripts asociados a a *Crisol* como aplicación y universo de juego. El archivo tendrá dos secciones las cuales estarán destinadas, precisamente, a especificar los scripts ya citados.

B.13.1. Sección [WorldEvents] .

En esta sección se establecerán los eventos asociados al universo de juego. Todos los eventos se especificarán utilizando la variable `ScriptEvent[x]` siendo `x` el identificador del evento para el que asociar un archivo script. Los posibles valores de `x` serán `OnWorldIdle`, para especificar el script asociado al evento idle del universo de juego, `OnNewHour`, para especificar el script asociado a una nueva hora, `OnStartGame`, para especificar el script asociado al evento de inicio de un nuevo juego y `OnEnterInArea` para especificar el evento asociado a la entrada en un área.

Ejemplo:

```
[WorldEvents]
 ScriptEvent[OnNewHour] = Script_OnNewHour.cst
 ScriptEvent[OnStartGame] = Script_OnStartGame.cst
 ScriptEvent[OnEnterInArea] = Script_OnEnterInArea.cst
End
```

B.13.2. Sección [GameEvents].

En esta sección se establecerán los eventos asociados a *Crisol* como aplicación. Todos los eventos se especificarán utilizando la variable `ScriptEvent[x]` siendo `x` el identificador del evento para el que asociar un archivo script. Los posibles valores de `x` serán `OnClickHourPannel`, para especificar el script asociado a la selección y pulsación del panel horario, `OnFleeCombat`, para especificar el script asociado a una nueva hora, `OnKeyPressed`, para especificar el script asociado al evento de pulsación de una tecla por parte del usuario, `OnStartCombatMode`, para indicar que se ha iniciado el modo combate y `OnEndCombatMode`, para indicar que ha finalizado el modo combate.

Ejemplo:

```
[GameEvents]
 ScriptEvent[OnClickHourPanel] = script_onclickhourpanel.cst
 ScriptEvent[OnKeyPressed] = script_onkeypressed.cst
End
```

B.14. El archivo “CrisolEngine.ini”.

Este archivo contendrá los parámetros de inicialización de *Crisol* como aplicación, es decir, desde aquí se definirán las configuraciones de los distintos subsistemas del motor sensibles de ser parametrizados por parte del diseñador.

Este archivo será también el más delicado y complicado desde el punto de vista de que los valores que aquí se establezcan podrán variar de forma notable el rendimiento del motor. Teniendo esto en cuenta, se recomendarán unos valores estándar que serán especificados en los ejemplos que se muestren para cada sección.

B.14.1. Sección [SysVar].

En esta sección se establecerán las variables del sistema. En la versión actual de *Crisol*, estos valores estarán orientados completamente al trabajo con el rendimiento de ejecución del bucle principal del motor.

Habrá que considerar dos variables. Por un lado tendremos la variable `AITickPerSecond`, que servirá para indicar cuantas veces en un segundo se producirá un “tick de IA”. Por “tick de IA” nos referiremos al instante en el que se recogerán los datos de la entrada y se ejecutarán. Lo normal será establecer un valor 30 ticks en 1 segundo. La otra de las variables será `NumMaxLoops` que servirá para indicar el número máximo de iteraciones que realizará. Este bucle al que nos referimos será el que controle el desfase de tiempo entre la etapa de dibujado de las escenas y los “ticks de IA”. En ordenares lentos el dibujado podrá llevar mucho tiempo y ese tiempo hará que se tenga que recuperar más tarde.

Ejemplo:

```
[SysVar]
 AITicksPerSecond = 30.0
 NumMaxLoops = 16
End
```

B.14.2. Sección [Timer].

En esta sección se establecerán los valores asociados al timer del sistema. Únicamente existirá una variable llamada `PerformanceOnlyFlag` cuyo objetivo será indicar si se desea que exista sólo el timer Performance que será el más avanzado de todos pero que en ciertos ordenadores más antiguos podría no existir. Lo ideal será poner que no se desea que sólo funcione *Crisol* con tal timer ya que de esta forma, especificamos que *Crisol* se adapte al mejor timer que encuentre en sistema.

Ejemplo:

```
[Timer]
 PerformanceOnlyFlag = false
End
```

B.14.3. Sección [FontSystem].

En esta sección se especificará el trabajo con el subsistema de fuentes. Inicialmente se deberá de establecer el nombre de la fuente por defecto usando la variable `DefaultFont`. Nótese que se deberá de especificar el nombre de una fuente True Type que sea común para la mayoría de equipos (por ejemplo, Arial o Times Roman) ya que de lo contrario, *Crisol* no podrá terminarse de ejecutar.

Una vez establecida la fuente por defecto, se podrán asociar otras con las que el motor podrá dibujar textos en pantalla. Para ello, se utilizará la variable `OptionalFont[x]` siendo `x` un valor entre 0 y el número máximo de fuentes opcionales menos uno. El único inconveniente será que en la versión actual de *Crisol* no se podrá cambiar de fuente y siempre será utilizada la que se establezca por defecto.

Ejemplo:

```
[FontSystem]
 DefaultFont = Arial
 OptionalFont[0] = Verdana
 OptionalFont[1] = Andy
 OptionalFont[2] = Adventure
End
```

B.14.4. Sección [GraphicSystem].

Esta sección será una de las más importantes pues será donde se especifique la inicialización del motor gráfico. En primer lugar se deberá de establecer el tipo de dispositivo (tarjeta gráfica) con el que se trabajará. Mediante `GraphicDeviceID` indicaremos el tipo de tarjeta con el que se trabajará. Si vale 0 se querrá indicar que será una tarjeta 2D/3D mientras que si vale 1, lo que se querrá indicar es que el equipo tendrá una tarjeta 2D y otra 3D por separado. Lo normal y estándar será especificar el valor 0. La otra variable relacionada será `3DDeviceType` que indicará el tipo de funcionamiento del dispositivo 3D. Si vale 0 indicaremos trabajo sobre una tarjeta gráfica TNT de la marca NVidia, si vale 1 sobre una tarjeta gráfica 3D general, si vale 2 indicaremos que se trabaje utilizando las extensiones MMX y, por último, si vale 3 se especificará trabajo vía software en exclusiva. Lo normal y estándar será especificar que queremos trabajar usando un dispositivo hardware general, esto es, lo normal será utilizar la opcion 1.

A continuación se indicará la resolución del dispositivo de vídeo y la profundidad del color. Para la resolución se utilizarán las variables `VideoWidth` y `VideoHeight` respectivamente. Se deberá de tener en cuenta que las resoluciones posibles serán 640x480, 800x600 y 1024x768 y que desde el principio el diseñador deberá de tener en cuenta este detalle pues la localización de los elementos de interfaz estarán basados en la resolución. Por último, indicar que la densidad en píxels se indicará utilizando la variable `VideoBpp`. Los valores posibles serán 16, 24 y 32 pero se recomendará utilizar siempre 16 ya que será siempre aceptado en todas las tarjetas de vídeo.

El siguiente paso irá asociado a especificar el número de buffers de dibujado a utilizar. Se especificará mediante `NumBackBuffers` y su valor ideal será de 1. En caso de disponer de una tarjeta gráfica con mucha memoria se podrá utilizar 2 y en caso de tener un dispositivo gráfico con poca memoria se podrá ahorrar de la misma estableciendo un valor de 0. Al no usar ningún buffer de intercambio el dibujado de las escenas será un poco mas brusco pero se dejará más espacio para alojar en memoria de vídeo elementos gráficos de uso durante lo que será el juego en sí.

El último paso será el de especificar la forma en la que se realizará la operación de volcado de las imágenes a mostrar. Se especificará mediante `VSynFlipFlag` y sus posibles valores serán `true` y `false`. Mediante true querremos indicar que el dibujado será sincronizado, esto es, se realizará de forma más lenta pero completamente suave. Especificando un valor de `false` podrá producirse un pequeño efecto de “cortinilla” pero se incrementarán considerablemente los frames por segundo. Lo recomendado será indicar que no se desea esperar por el sincronizado vertical, es decir, se recomendará especificar un valor de false.

Ejemplo:

```
[GraphicSystem]
 GraphicDeviceID = 0
 3DDeviceType = 1
 VideoWidth = 640
 VideoHeight = 480
 VideoBpp = 16
 NumBackBuffers = 1
 VSyncFlipFlag = false
End
```

B.14.5. Sección [AudioSystem].

En la sección de audio se especificará la forma en la que será inicializado el dispositivo de audio (la tarjeta de sonido). Para ello, existirán tres variables básicas. Por un lado tendremos `Frequency`, que indicará la frecuencia a la que se deserá inicializar el dispositivo. Los valores posibles serán 11, 22 y 44, siendo 11 la menor calidad y 44 la mayor. Por otro `NumChannels`, que se utilizará para indicar el número de canales a utilizar. Si se especifica 1 se querrá indicar una reproducción mono y si vale 2 estéreo. Por último, mediante la variable `Bpm` querremos indicar los bits que tendrán las muestras (archivos de sonido WAV). Los dos únicos posibles valores serán 8 y 16, siendo 8 la menor calidad de todas.

Cabe tener en cuenta que a menor calidad mas ahorro en recursos y a mayor calidad, pese a que obtendremos una mejor reproducción de los sonidos, se consumirán más recursos del sistema. De todas formas, se recomienda utilizar la mejor calidad siempre que sea posible.

Ejemplo:

```
[AudioSystem]
 Frequency = 44
 NumChannels = 2
 Bpm = 16
End
```

B.14.6. Sección [GameDataBase].

Esta será la última sección. Estará destinada a configurar el módulo destinado a manejar el acceso a los archivos de datos "`.cbb`" y "`.cbt`" con los que trabajará el motor. Habrá cuatro partes fundamentales a comentar que serán, la especificación de los archivos "`.cbb`", la especificación de los archivo "`.cbt`", la especificación del posible path donde se hallen los ficheros de áreas, la especificación del nombre del archivo de scripts globales y, finalmente, la especificación del nombre del archivo con la *TileRegionMaks* (que será un gráfico suministrado con *Crisol* para ayudar al motor isométrico en procesos internos).

Los archivos "`.cbb`" se especificará utilizando, siempre, el prefijo `Files.CBB.`. A este prefijo se le podrán añadir la terminación `AnimTemplates`, para especificar el archivo de perfiles de animación, `CriaturesProfiles`, para indicar el archivo de perfiles de criaturas, `WallProfiles`, para indicar los perfiles de paredes, `ItemProfiles`, para indicar el archivo de perfiles de ítems, `RoofProfiles`, para indicar los perfiles de los tiles que representen techos, `SceneObjProfiles`, para indicar el archivo de perfiles de objetos de escenario y `FloorProfiles`, para indicar el nombre del archivo con los tiles que representen el suelo de las áreas. Todos los archivos deberán de ser especificados sin su extensión "`.cbb`".

Para trabajar con los archivos con el diseño de áreas se utilizarán ficheros con nombre "`areax.cbb`" donde x será un identificador entre 0 y el número máximo de áreas diseñadas menos uno. Como el acceso será automático, se dará la oportunidad al diseñador para especificar un valor de path o ruta de acceso donde pudieran estar. Por defecto no habrá ruta de acceso pero de haberlo, se deberá de indicar terminando con el carácter '\'. La variable utilizada para recoger esta información será `Files.CBB.AreaFilesPath`.

El siguiente paso será especificar los nombres de los archivos "`.cbt`". Para indicarlos se utilizarán las variables cuyo prefijo sea `Files.CBT.`. Así, se utilizará la terminación `Rules` para especificar el archivo de reglas, `StaticText`, para especificar el nombre del archivo con el texto estático, `PlayerProfile`, para especificar los perfiles del jugador, `GFXDefs`, para especificar el archivo de GFX, `PresentationProfiles`, para indicar el nombre del archivo de perfiles de presentaciones, `InterfacesCfg`, para especificar el nombre de configuración de los interfaces y `GlobalScriptEvents` para hacer lo propio con el archivo de scripts globales. En todos los casos, no se deberá de escribir la extensión "`.cbt`" de los archivos especificados.

A continuación se indicará el nombre del archivo con los scripts compilados (si es que hay scripts). No se tendrá que poner la extensión del mismo, pues el motor automáticamente buscará por la terminación "`.cst`". Para establecer este valor se utilizará la variable `Files.Scripts`.

El último paso consistirá en indicar el nombre del archivo con la *TileRegionMask*. Bastará con poner el nombre (con la extensión) del archivo tga que contiene dicho gráfico especial. Se usará la variable `Files.TileRegionMask`.

Ejemplo:

```
[GameDataBase]
 Files.CBT.Rules = Rules
 Files.CBT.StaticText= StaticText
 Files.CBT.PlayerProfiles = PlayerProfiles
 Files.CBT.GFXDefs = GFXDefs
 Files.CBT.PresentationProfiles = PresentationProfiles
 Files.CBT.InterfacesCfg = InterfacesCfg
 Files.CBT.GlobalScriptEvents = GlobalScriptEvents
 ; Recordar que esta variable no será obligatoria
 Files.CBB.AreaFilesPath = Areas\
 Files.CBB.AnimTemplates = AnimTemplates
 Files.CBB.CriatureProfiles = CriatureProfiles
 Files.CBB.WallProfiles = WallProfiles
 Files.CBB.ItemProfiles = ItemProfiles
 Files.CBB.RoofProfiles = RoofProfiles
 Files.CBB.SceneObjProfiles = SceneObjProfiles
 Files.CBB.FloorProfiles = FloorProfiles
End
```

B.15. Utilización de CrisolBuilder.

CrisolBuilder es una utilidad que tiene dos objetivos principales, por un lado chequear todos los archivos "`.cbt`" explicados anteriormente y por otro pasar a formato binario los archivos de perfiles de animaciones y de entidades así como los que definen las distintas áreas. Estos archivos serán pasados a binario para incrementar la velocidad de proceso ya que la lectura de un archivo de texto lleva más tiempo.

Su utilización es muy sencilla pues sólo habrá que configurar un archivo llamado "`CrisolBuilder.ini`" (que tiene la misma configuración de secciones que un archivo "`.cbt`"), y ejecutar la utilidad teniendo, eso sí, todos los archivos "`.cbt`" creados. *CrisolBuilder* se encargará de mostrar los mensajes de error o de advertencia oportunos para indicar dónde ha localizado problemas en los archivos "`.cbt`".

B.15.1. El archivo "CrisolBuilder.ini".

Este archivo tendrá como objetivo principal especificar los nombres de los diferentes archivos "`.cbt`" así como el número de áreas que *CrisolBuilder* deberá de procesar. Siempre será leído por la utilidad nada más que ésta arranque. Ni que decir tiene que si no está creada o existe pero se ha editado de forma incorrecta, no se podrá ejecutar la utilidad.

En la sección `[CBTFileNames]` se especificarán en la variable `RulesDefsFileName`, el nombre del archivo de reglas, en `FloorProfilesFileName` el nombre del archivo de perfiles floor, en `WallProfilesFileName` el nombre del archivo de perfiles de pared, en `SceneObjProfilesFileName` el nombre del archivo de perfiles de objetos de escenario, en `ItemProfilesFileName` el nombre del archivo de perfiles de ítems, en `CriatureProfilesFileName` el archivo de pefiles de criaturas no jugador (npc), en `RoofProfilesFileName` el archivo de las entidades techo, en `PlayerProfilesFileName` el archivo de perfiles de jugador y la forma en que el usuario podrá crear a su héroe cuando comience una nueva partida, en `InterfacesCfgFileName` el nombre del archivo con la configuración de los interfaces, las plantillas de animación estarán en el archivo cuyo nombre se asociará a la variable `AnimTemplatesFileName`, el texto estático será leido del archivo cuyo nombre se especificará en `StaticTextFileName`, en `GFXDefsFileName` se especificará el nombre del archivo de perfiles GFX y en `PresentationsFileName` el nombre del archivo de los perfiles de presentaciones.

Por último, se deberá de incluir el número de áreas a procesar usando la variable `NumberOfAreas`, como al menos deberá de existir un área, su valor deberá de ser al menos uno. Para el trabajo con los archivos de área se utilizará este formato porque estos archivos tendrán su nombre basado siempre en su identificador.

Después de definir esta sección, se deberá de trabajar con la sección `[Params]`, que tendrá como objetivo especificar los parámetros de ejecución que permitirarán modificar en un determinado grado la ejecución de la utilidad.

El primer parámetro estará enfocado a indicar si queremos que los mensajes salgan por pantalla o por disco. Para establecer lo que queremos al respecto usaremos la variable `WriteMsgInFileFlag`. Su valor por defecto será de `false` así que de no poner nada los mensajes se enviarán a la pantalla. En caso de que valga `true`, *CrisolBuilder* creará el archivo "`CBOutput.txt`" donde guardará los mensajes.

Los siguientes parámetros estarán relacionados a indicar si queremos que se muestren los perfiles que han sido escogidos. En todos los casos, los valores por defecto serán de false, indicando que no queremos que se escriba información alguna. Todas las variables comenzarán por `ReportProfilesUsed.` de tal forma que la terminación `AnimTemplates` servirá para indicar si queremos que se muestren los perfiles de plantillas de animación utilizados, `Floors` para los perfiles de entidades suelo, `Roofs` para los perfiles de las entidades techo, `Items` para los perfiles de ítems, `Walls` para los perfiles pared,

`SceneObjects` para los perfiles de objetos de escenario y `Criatures` para los perfiles de criaturas. Como toda esta información ocupará muchas pantallas en caso de mostrar la información en pantalla, lo más normal será que, en caso de querer recabar la información en algunos de estos casos, se habilite el envio de información a disco.

Ejemplo:

```
[CBTFileNames]
 RulesDefsFileName = Rules
 NumberOfAreas = 2
 FloorProfilesFileName = FloorProfiles
 WallProfilesFileName = WallProfiles
 SceneObjProfilesFileName = SceneObjProfiles
 ItemProfilesFileName = ItemProfiles
 CriatureProfilesFileName = CriatureProfiles
 RoofProfilesFileName = RoofProfiles
 PlayerProfilesFileName = PlayerProfiles
 InterfacesCfgFileName = InterfacesCfg
 AnimTemplatesFileName = AnimTemplates
 StaticTextFileName = StaticText
 GFXDefsFileName = GFXDefs
 PresentationsFileName = PresentationProfiles
End

[Params]
 WriteMsgInFileFlag = true
 ReportProfilesUsed.AnimTemplates = false
 ReportProfilesUsed.Floors = false
 ReportProfilesUsed.Roofs = true
 ReportProfilesUsed.Items = true
 ReportProfilesUsed.Walls = false
 ReportProfilesUsed.SceneObjects = false
 ReportProfilesUsed.Criatures = false
End
```

B.15.2. Tipos de mensajes.

Básicamente existirán 3 tipos de mensajes. Por un lado tendremos los referidos a errores, que se mostrarán cuando se asocie un valor no válido a una variable y harán que la ejecución termine o inmediatamente o después de finalizar el reconocimiento sobre la sección actual (independientemente del archivo "`.cbt`" que se esté tratando). Luego estarán los mensajes de error fatal que aparecerán siempre que se olvide establecer una sección o una variable sin valor por defecto y, por último, estarán los mensajes de advertencia que simplemente advertirán algún tipo de situación que no genera error pero que puede producir confusiones si dicha situación se ha producido por despiste del diseñador.