UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

"CraftCosta: Creación de un plug-in MMORPG para servidores de Minecraft"

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Valldecabres Barrera, Julián

Tutor: Mollá Vayá, Ramón Pascual

Valencia 2016

RESUMEN

Desarrollo de un complemento para Minecraft que transforme un servidor convencional de Minecraft, en un servidor MMORPG, extendiendo y haciendo uso de la API de Minecraft (Bukkit) y aportando una interfaz gráfica para el administrador del juego.

Palabras clave:

API Bukkit, Minecraft, complemento, Java, MMORPG, clanes, sistema de niveles, sistema de experiencia, sistema de monstruos, comercio, chat.

ABSTRACT

Development of a plugin to transform a conventional Minecraft server into a MMORPG server, extending and using the Minecraft API (Bukkit) and providing a graphical interface for the game administrator.

Linked words:

API Bukkit, Minecraft, plugin, Java, MMORPG, guilds, leveling system, experience system, monster system, trade, chat.

Tabla de contenido

1 Introducción

1.1 Motivación

Desde una edad temprana tuve interés por el mundo de los videojuegos y de la informática. A la edad de 10 años tuve mi primera consola, una Mega drive[1] de la compañía SEGA[2], con la cual, solo jugaba 2 horas cada fin de semana, además de realizar cursillos de verano para aprender cómo manejar un ordenador y mi primer lenguaje de programación BASIC[3].

A los 11 años obtuve mi primer ordenador clónico, en aquel entonces era el único que mi familia podía permitirse y a pesar de ser de muy baja gama siempre hice uso de él eficientemente. Mi padre, que me acompañaba en mis horas de videojuego y sabiendo la actitud que mostraba ante la informática me compró una suite de diseño de videojuegos llamada DIV Games Studio[4], con el cual llegué a la conclusión que no tenía la suficiente base para la creación de los mismos, y es por ello, que tras pasar la secundaria hice un ciclo superior de desarrollo de aplicaciones informáticas.

Si bien, mi interés en la creación de videojuegos decayó un poco en aquellos años que transcurrieron en plena crisis económica, decidí realizar el grado en ingeniería informática y en esa misma fecha también comencé mi andanza en el videojuego llamado Minecraft[5].

Actualmente administro junto con un compañero de grado un servidor de Minecraft, que hace uso de más de 80 complementos desarrollados de manera libre por la comunidad de desarrolladores, y hasta ahora no hemos creado más que complementos de administración para la mejora de la eficiencia de nuestro servidor. Con este proyecto pretendo desarrollar un complemento que aporte un valor añadido a mi servidor y que de manera libre pueda ser compartido entre la comunidad de desarrolladores de complementos de Minecraft y los administradores que hacen uso de ellos.

1.2 Objetivos

A continuación se detallarán los principales objetivos que se pretenden conseguir con la realización de este TFG.

El objetivo principal pretende la realización de un complemento altamente configurable para la gestión de un servidor MMORPG[6] con los elementos básicos que los video

---

[1] Mega drive: Consola de la empresa SEGA de 16-bits, también llamada Génesis en otras partes del mundo.

[2] SEGA: Compañía desarrolladora de videojuegos y consolas. http://www.sega.com/

[3] BASIC: Lenguaje de programación de propósito general para principiantes. https://en.wikipedia.org/wiki/BASIC

[4] DIV Games Studio: Motor de desarrollo de videojuegos lanzado en 1998 por la empresa española Hammer Technologies.

[5] Minecraft: Videojuego indie de supervivencia desarrollado por Mojang AB y comprado por Microsoft. https://www.minecraft.net/es/

[6] MMORPG: Videojuego de rol multijugador masivo.

juegos de esta índole están compuestos o en este caso una mayoría, aunándolos en un mismo complemento, para la mejora de la eficiencia, reducir la carga de trabajo del servidor disminuyendo el número de complementos con los que un servidor de Minecraft, de esta índole, suelen manejar.

En segundo lugar se pretende mejorar el modo de configuración de los complementos que hasta ahora existen (ficheros de texto plano Yaml[7]), proporcionando una interfaz para los administradores, lo más agradable posible, para la configuración del complemento, haciendo uso del potencial de Java[8] para que un mismo ejecutable sea a la vez un complemento, en este caso de gestión y configuración, y un complemento ejecutable en el entorno de un servidor no oficial de Minecraft.

Finalmente, se pretende demostrar el alto grado de flexibilidad de este juego, mostrando cuanto puede llegar a modificar las funcionalidades básicas que trae por defecto para conseguir un juego de rol, por ejemplo, modificando el comportamiento natural de los monstruos y las habilidades de los jugadores, mediante armas y armaduras modificadas.

1.3 Estructura de la obra

Inicialmente se comentará sobre el estado del arte de Minecraft, con una breve explicación de sus modos de juego y más detenidamente sobre el modo online del mismo, de manera extendida sobre los complementos de los servidores de Minecraft que son el propósito de este TFG, y seguido un análisis del porque ha sido esta la elección para realizar este TFG.

Siguiendo al estado del arte se continuará con una crítica del mismo haciendo hincapié en las carencias que encontradas tanto en el desarrollo de este TFG como en el tiempo disfrutado en Minecraft, tanto administrando un servidor como jugando.

A continuación de la crítica se realizará una propuesta que intentará salvar las carencias anteriormente citadas en la crítica al estado del arte seguido del análisis del problema en el que se detallarán los aspectos más importantes a tratar en este proyecto, entre ellos, requisitos del problema tanto de la aplicación como del complemento a desarrollar, un análisis de las herramientas tanto las provistas por Minecraft como las provistas por libros, foros u otras fuentes de ayuda, internacionalización del complemento, eficiencia algorítmica importante en estos tipos de juegos dado que manejan una ingente cantidad de información.

Se seguirá con el presupuesto en el que se detallarán los recursos tanto materiales como temporales necesarios para el desarrollo del proyecto, seguido de un diseño de la solución realizado mediante casos de uso tanto sobre la aplicación de gestión de escritorio para los administradores como del complemento en sí, realizando una aproximación a la estructura final de la aplicación en su totalidad. En el apartado de implementación se hará constancia de las partes de mayor dificultad en el desarrollo de este TFG y se mostrarán los resultados para posteriormente realizar un análisis exhaustivo sobre ellos.

Se continuará con las conclusiones obtenidas al finalizar el desarrollo de la solución, una breve relación entre el proyecto y los estudios cursados en la carrera nombrando que conocimientos o tecnologías eran requeridos y en cuales se hubiera necesitado hacer hincapié o abordar nuevos conocimientos, una lista de posibles trabajos futuros,

---

[7] YAML: Formato de serialización de datos legible http://yaml.org/

[8] Java: Lenguaje de programación orientado a objetos https://www.oracle.com/es/index.html

mejoras o ampliaciones aplicables sobre este proyecto o de la misma índole.

Por último se podrá encontrar la bibliografía, los anexos en los cuales se hará explicaciones extensas del mundo que es Minecraft y el glosario de términos donde se podrá acudir para aclarar que significan ciertos términos propios del ámbito en el que se desarrolla el proyecto.

1.4 Convenciones

En este apartado se detallarán las convenciones o acuerdos con el lector:

- Cada vez que aparezca una palabra en otro idioma, normalmente será en inglés, esta será escrita en cursiva, su primera aparición tendrá al pie de página una descripción y si es necesario una explicación extensa en el glosario, mediante un enlace desde el pie de página.
- Las siglas estarán en mayúsculas y en el caso que sea necesario en su primera aparición al pie de página tendrá una breve explicación y más extensamente explicada en el glosario.
- La primera vez que aparezcan los nombres de empresas o productos tendrán al pie de página una breve descripción y/o un enlace a la web.
- Nombres propios del ámbito de Minecraft (entidades, objetos) estarán acompañados de una traducción, si es posible, entre paréntesis.
- La primera aparición de nombres propios de personas estarán acompañados de su apodo si lo tienen, el resto de apariciones se hará uso de su apodo.
- Los nombres de clases o eventos que hacen referencia a partes de código del proyecto estarán escritas en cursiva y negrita.

2 Estado del arte

2.1 Minecraft

Minecraft es un videojuego de tipo supervivencia y construcción desarrollado en el lenguaje de programación Java. Se caracteriza por hacer uso de cubos que representan distintos materiales con los que el jugador puede interactuar para crear construcciones, en una versión minimalista de la realidad. Incluye entidades pasivas o agresivas con las que el jugador puede obtener recursos, entre las pasivas (gallinas, cerdos, vacas, etcétera) se pueden obtener alimentos, cueros u otros materiales y entre las agresivas (esqueletos, *Creepers*[9], zombis) materiales y orbes de experiencia. También el jugador puede realizar los llamados *Craftings*[10], que consiste en la combinación de distintos materiales para obtener nuevos materiales y/o utensilios. La mezcla de los elementos citados anteriormente aporta una libertad al juego que solo es acrecentada por los modos de juego, entre ellos el modo multijugador. (Véase Anexo I para una descripción más detallada)

[9] Creepers: monstruo de Minecraft característico de este videojuego, por provocar sustos a sus jugadores, se puede leer una descripción más detallada en el Anexo I.

[10] Craftings o elaboraciones, son la combinación de varios objetos para la obtención de otros.

2.1.1 Servidores

Los servidores de Minecraft se pueden diferenciar entre los que son oficiales y los no oficiales. Los servidores oficiales reciben el nombre de *Vanilla*[11] y se caracterizan por su modo de juego de supervivencia es exactamente igual al modo de juego un solo jugador del videojuego original salvo que es multijugador, su principal desventaja es que no acepta complementos y por tanto no son extensibles en el sentido de añadir nuevas funcionalidades. Por otro lado, se encuentran las versiones no oficiales de los servidores de Minecraft, que son los usados en este proyecto. A continuación una breve explicación de los servidores no oficiales llamados comúnmente *mods*[12].

Los mods, son modificaciones no oficiales del servidor de Minecraft, con ellas es posible añadir complementos que añaden nuevas funcionalidades, entre los mods más conocidos se encuentran: *CraftBukkit*[13]*, Spigot*[14] y *Sponge*[15], entre otros. La razón de su existencia es cubrir la carencia que tiene el servidor oficial de Minecraft, que es la de poder extender su funcionalidad e incluso mejorar su rendimiento. Estas modificaciones son reemplazos completos del servidor oficial de Minecraft, desarrollados por equipos de desarrolladores que empezaron a jugar Minecraft y siguieron creando su propia modificación del servidor oficial en una edad temprana de desarrollo, este proyecto que inició la revolución de sus distintas versiones modificadas del servidor oficial se llamó hMod Project[16]. (Véase Anexo II para una descripción más detallada de los mods de Minecraft)

2.1.2 Complementos

Los complementos, en inglés *plugins*, que es en lo que se basa y el propósito de este proyecto, son desarrollados mediante unas API[17] (no oficial) y añaden a los servidores nuevas funcionalidades o modifican las ya existentes, cabe destacar que los complementes tienden a ser dependientes de la versión del servidor en el que están alojados por lo que es importante mantener la coherencia entre la versión del servidor y los complementos. Cada mod tiene su propia API por lo que es necesario saber qué tipo de mod y API usar para desarrollar un complemento nuevo.

En la actualidad existe gran variedad de complementos tanto de gestión interna del servidor, como puede ser: la gestión de usuarios, creación de mundos o aspectos de rendimiento como la gestión de entidades activas, como de adición de funcionalidades tales como: sistemas monetario, mini-juegos, efectos visuales que mejoran el entorno visual del videojuego en sí. (Véase Anexo II para una descripción más detallada de los complementos de Minecraft)

---

[11] Vanilla: Nombre que reciben los servidores Minecraft oficiales.

[12] Mods: Nombre que reciben los servidores modificados de Minecraft.

[13] CraftBukkit: Nombre que reciben los servidores modificados por el proyecto Bukkit. https://bukkit.org/

[14] Spigot: Nombre que reciben los servidores modificados por el proyecto SpigotMC. https://www.spigotmc.org/

[15] Sponge: Nombre que reciben los servidores modificados por el proyecto Sponge. https://www.spongepowered.org/

[16] HMod Project: Proyecto que inició los mods para Minecraft. https://www.openhub.net/p/hmod

[17] API: Interfaz de programación de aplicaciones o librería.

2.2 Porqué Minecraft

Minecraft desde su lanzamiento al mercado ha tenido un gran auge, por su simplicidad y sus infinitas posibilidades de creación, y en mi opinión parte de ese auge ha sido por la colaboración de los usuarios en el proyecto que es Minecraft en sí, haciéndolos participes de un todo. A continuación relataré mis razones de porqué elegí desarrollar este proyecto de manera libre.

En principio, cuando empecé la carrera, Minecraft fue uno de mis entretenimientos favoritos, no por ello elegí realizar este proyecto, pero influenció bastante ya que en Febrero de 2014 un compañero de carrera y yo montamos un servidor, nos gustó la experiencia de administrar y gestionar a los jugadores que entraban. Al poco tiempo ambos empezamos a crear nuevos complementos y modificarlos, nuestros y de otros creadores respectivamente, para mejorar la experiencia de juego o actualizarlos cuando los propios creadores no lo hacían. En nuestro servidor siempre fantaseaba con poder realizar un mundo exclusivamente de rol y siempre estaba retrasándolo por la carrera, así que decidí aprovechar y hacerlo con la excusa del TFG.

Por otro lado, el conocer Java y todo el entramado detrás de las modificaciones no oficiales del servidor supuso una facilidad para esta elección, por supuesto ni conozco todo el potencial de Java, ni todos los recovecos de Minecraft, pero en sí es un desafío que sí me gustaría realizar. El poder acceder a los complementos públicos y ver como desarrollaban los plugins me aportó ideas para el propio proyecto, que podría usar para mejorar, con mis conocimientos adquiridos en la carrera, la eficiencia y nuevas funcionalidades del proyecto.

Por último, la cantidad de información que se puede encontrar por: foros, libros y enciclopedias digitales, a mi parecer es la necesaria para el desarrollo del proyecto.

3 Critica al estado del arte

En la siguiente crítica se pondrán de manifiesto las carencias que se encuentran en los servidores no oficiales de Minecraft.

Los servidores de Minecraft basados en rol suelen estar compuestos por una amalgama de complementos que en ocasiones son inconexos dado que suelen ser de distintos desarrolladores y no existen estándares de implementación, de modo que cada complemento puede tratar distintos aspectos de un videojuego de rol (cómo puede ser un complemento dedicado explícitamente a habilidades, creación de artículos especiales, clanes, misiones y un gran etcétera), y es difícil realizar una combinación de complementos para llegar a una solución estable y eficiente en un servidor. Existen servidores de rol que sí usan una suite completa, pero no se distribuye de manera libre, por lo que no es accesible a todo el mundo.

La configuración de los complementos que componen un servidor de Minecraft dedicado al rol se realiza sobre ficheros de texto plano *Yaml*[18], o dentro del juego, lo que dificulta una posible visión global de la configuración de todos los complementos dedicados al rol del mismo servidor.

Los servidores de Minecraft limitan en gran medida los atributos de las entidades del juego, no permitiendo así la posibilidad de modificación. Existen complementos que

[18] Yaml (Yaml another markup language): es un formato de serialización de datos para cualquier tipo de lenguaje de programación. http://yaml.org/

modifican estos aspectos de las entidades mediante la captura de eventos lo cual disminuye la eficiencia del servidor y limita su rendimiento.

4 Propuesta

Se pretende desarrollar un complemento que sea una suite de los elementos de los que, habitualmente, se compone un servidor de rol, altamente configurables y que pueda ser distribuido de manera libre.

Mejorar el sistema de configuración de complementos en general, no solo aportando una interfaz gráfica de usuario dedicada a los administradores de los servidores, sino además agilizando la metodología de instalación y configuración de estos complementos, reduciendo los pasos a: crear una configuración global e iniciar el servidor.

Extender la funcionalidad de entidades, de objetos y de los jugadores para aumentar las posibilidades en Minecraft, creando entidades altamente configurables y medios por los que el jugador pueda mejorarse tanto en atributos como en equipamiento.

5 Análisis del problema

A continuación se detallaran aspectos globales, que ayudarán al desarrollo de la solución. Se comenzara con un análisis de requisitos que tiene como fin detallar las funcionalidades finales que la solución tendrá, basada en casos de uso y acompañado de prototipos si compete, seguido de un análisis de las herramientas que se harán uso y los aspectos de seguridad, eficiencia algorítmica e internacionalización que se tendrán en cuenta.

5.1 Análisis de requisitos

La solución se divide en dos partes diferenciadas, por un lado, la aplicación con la que los administradores podrán crear una configuración concreta para un servidor, y por otro lado, el complemento que modificará las funcionalidades del servidor en función de la configuración creada por defecto o configurada por los administradores a priori. Cabe destacar que la aplicación y el complemento estarán incluidos en un mismo ejecutable para dar portabilidad a la solución.

5.1.1 Definición de actores

Los actores serán los tipos de usuarios que interactuarán tanto con la aplicación de escritorio, como con el complemento instalado en el servidor.

Referencia	ACT-01
Nombre	Administrador del servidor
Comentarios	Interviene en la aplicación y el complemento

Referencia	ACT-02

Nombre	Jugador del servidor
Comentarios	Interviene solo en el complemento

5.1.2 Definición de objetivos

En este apartado definiré los diferentes objetivos que se esperan alcanzar cuando finalice este proyecto. Serán especificados mediante una plantilla de objetivos.

Referencia	OBJ-01
Nombre	Gestionar jugadores
Descripción	El sistema deberá gestionar a los jugadores
Comentarios	

Referencia	OBJ-02
Nombre	Gestionar clases
Descripción	El sistema deberá gestionar las clases a los que los jugadores podrán acceder
Comentarios	

Referencia	OBJ-03
Nombre	Gestionar niveles
Descripción	El sistema deberá gestionar los niveles de los jugadores
Comentarios	

Referencia	OBJ-04
Nombre	Gestionar chat
Descripción	El sistema deberá gestionar los mensajes de los jugadores en el servidor.
Comentarios	

Referencia	OBJ-05
Nombre	Gestionar clanes
Descripción	El sistema deberá gestionar los clanes existentes en el servidor
Comentarios	

Referencia	OBJ-06
Nombre	Gestionar grupos
Descripción	El sistema deberá gestionar los grupos de jugadores en el servidor
Comentarios	

Referencia	OBJ-07
Nombre	Gestionar armaduras
Descripción	El sistema deberá gestionar las armaduras dentro del servidor.
Comentarios	

Referencia	OBJ-08
Nombre	Gestionar armas
Descripción	El sistema deberá gestionar las armas dentro del servidor.
Comentarios	

Referencia	OBJ-09
Nombre	Gestionar joyas
Descripción	El sistema deberá gestionar las joyas dentro del servidor.
Comentarios	

Referencia	OBJ-10
Nombre	Gestionar objetos
Descripción	El sistema deberá gestionar los objetos dentro del servidor.
Comentarios	

Referencia	OBJ-11
Nombre	Gestionar la economía
Descripción	El sistema deberá gestionar la economía de los jugadores en el servidor
Comentarios	

Referencia	OBJ-12
Nombre	Gestionar monstruos
Descripción	El sistema deberá gestionar los monstruos en el servidor
Comentarios	

Referencia	OBJ-13
Nombre	Gestionar generadores de monstruos o *spawners*[19]
Descripción	El sistema deberá gestionar los generadores de monstruos en el servidor.
Comentarios	

Referencia	OBJ-14
Nombre	Gestionar configuración general.
Descripción	El sistema deberá gestionar las configuraciones con la aplicación de escritorio.
Comentarios	

5.1.3 Requisitos de almacenamiento de información

Este apartado contiene la lista de requisitos de almacenamiento de información identificados mediante una plantilla.

Referencia	RI-01
Nombre	Información sobre jugadores
Objetivos asociados	OBJ-01 Gestión de jugadores OBJ-02 Gestión de clases OBJ-03 Gestión de niveles OBJ-04 Gestión de chat OBJ-05 Gestión de clanes OBJ-06 Gestión de grupos OBJ-07 Gestión de armaduras OBJ-08 Gestión de armas OBJ-09 Gestión de joyas OBJ-11 Gestión de economía

[19] Spawner es el nombre que reciben los generadores de monstruos en Minecraft, representados mediante un bloque en forma de jaula que contiene a la entidad que aparecerá en la localización en la que se encuentre.

Requisitos asociados	• CU-01 Conectar con el servidor • CU-02 Crear/Carga de datos del jugador • CU-03 Abandonar servidor • CU-04 Guardar datos del jugador • CU-05 Visualizar datos del jugador • CU-06 Seleccionar clase • CU-12 Subir de nivel • CU-18 Crear clan • CU-19 Unirse a clan • CU-20 Abandonar clan • CU-25 Crear grupo • CU-26 Unirse a grupo • CU-27 Abandonar grupo • CU-32 Poner/quitar armadura • CU-33 Mejorar armadura • CU-38 Poner/quitar arma • CU-39 Mejorar arma • CU-44 Poner/quitar joya • CU-45 Combinar joya • CU-50 Dar/Recibir dinero
Descripción	El sistema deberá almacenar la información correspondiente a los jugadores, en concreto:
Datos específicos	• UUID (Identificador único) • Nombre • Clase o profesión a la que está asociado • Atributos propios de la clase • Atributos derivados del equipo(armadura, arma, joyas) • Dinero • Clan asociado • Grupo asociado • Nivel • Experiencia
Comentarios	

Referencia	RI-02
Nombre	Información sobre clases
Objetivos asociados	OBJ-01 Gestión de jugadores OBJ-02 Gestión de clases
Requisitos asociados	• CU-06 Seleccionar clase

	• CU-07 Visualizar información de clases • CU-17 Crear clase • CU-18 Modificar clase • CU-19 Eliminar clase • CU-20 Habilitar/Deshabilitar clases
Descripción	El sistema deberá almacenar la información correspondiente a las clases, en concreto:
Datos específicos	• Nombre de la clase • Atributos de la clase
Comentarios	

Referencia	RI-03
Nombre	Información sobre niveles
Objetivos asociados	OBJ-01 Gestión de jugadores OBJ-03 Gestión de niveles
Requisitos asociados	• CU-12 Subir de nivel • CU-13 Crear/Modificar configuración de niveles • CU-14 Visualizar configuración de niveles.
Descripción	El sistema deberá almacenar la información correspondiente a los niveles, en concreto:
Datos específicos	• Relación niveles experiencia • Puntos de habilidad por nivel
Comentarios	

Referencia	RI-04
Nombre	Información sobre chat
Objetivos asociados	OBJ-04 Gestión de chat OBJ-05 Gestión de clanes OBJ-06 Gestión de grupos
Requisitos asociados	• CU-15 Usar chat • CU-16 Habilitar/Deshabilitar chat • CU-17 Crear/Modificar configuración de chats
Descripción	El sistema deberá almacenar la información correspondiente a los chats, en concreto:
Datos específicos	• Prefijos de chat • Símbolos de chat • Colores de chat

Comentarios	

Referencia	RI-05
Nombre	Información sobre clanes
Objetivos asociados	OBJ-01 Gestión de jugadores OBJ-05 Gestión de clanes
Requisitos asociados	• CU-18 Crear clan • CU-19 Unirse a clan • CU-20 Abandonar clan • CU-21 Invitar a clan • CU-22 Cambiar propietario del clan • CU-23 Crear/Modificar configuración de clanes • CU-24 Habilitar/Deshabilitar clanes
Descripción	El sistema deberá almacenar la información correspondiente a los clanes y su configuración, en concreto:
Datos específicos	• Nombre de clanes • Atributos del clan • Información de miembros • Configuración de clanes
Comentarios	

Referencia	RI-06
Nombre	Información sobre grupos
Objetivos asociados	OBJ-01 Gestión de jugadores OBJ-06 Gestión de grupos
Requisitos asociados	• CU-25 Crear grupo • CU-26 Unirse a grupo • CU-27 Abandonar grupo • CU-28 Invitar a grupo • CU-29 Cambiar propietario del grupo • CU-30 Crear/Modificar configuración de grupos • CU-31 Habilitar/Deshabilitar grupos
Descripción	El sistema deberá almacenar la información correspondiente a los grupos, en concreto:
Datos específicos	• Nombre de grupos • Atributos de grupos • Información de miembros

	• Configuración de grupos
Comentarios	

Referencia	RI-07
Nombre	Información sobre armaduras
Objetivos asociados	OBJ-01 Gestión de jugadores OBJ-07 Gestión de armaduras
Requisitos asociados	• CU-32 Poner/quitar armadura • CU-33 Mejorar armadura • CU-34 Crear set de armadura • CU-35 Modificar set de armadura • CU-36 Eliminar set de armadura • CU-37 Crear/Modificar configuración de armaduras
Descripción	El sistema deberá almacenar la información correspondiente a las armaduras, en concreto:
Datos específicos	• Nombre de armaduras • Atributos de armaduras • Configuración de armaduras
Comentarios	

Referencia	RI-08
Nombre	Información sobre armas
Objetivos asociados	OBJ-01 Gestión de jugadores OBJ-08 Gestión de armas
Requisitos asociados	• CU-38 Poner/quitar arma • CU-39 Mejorar arma • CU-40 Crear arma • CU-41 Modificar arma • CU-42 Eliminar arma • CU-43 Crear/Modificar configuración de armas
Descripción	El sistema deberá almacenar la información correspondiente a las armas, en concreto:
Datos específicos	• Nombre de armas • Atributos de armas • Configuración de armas
Comentarios	

Referencia	RI-09
Nombre	Información sobre joyas
Objetivos asociados	OBJ-01 Gestión de jugadores OBJ-09 Gestión de joyas
Requisitos asociados	• CU-44 Poner/quitar joya • CU-45 Combinar joyas • CU-46 Crear joya • CU-47 Modificar joya • CU-48 Eliminar joya • CU-49 Crear/Modificar configuración de joyas
Descripción	El sistema deberá almacenar la información correspondiente a las joyas, en concreto:
Datos específicos	• Nombre de joyas • Atributos de joyas • Configuración de joyas
Comentarios	

Referencia	RI-10
Nombre	Información sobre objetos
Objetivos asociados	OBJ-10 Gestión de objetos
Requisitos asociados	
Descripción	El sistema deberá almacenar la información correspondiente a los objetos, en concreto:
Datos específicos	• Nombre de objetos • Descripción de los objetos
Comentarios	

Referencia	RI-11
Nombre	Información sobre economía
Objetivos asociados	OBJ-10 Gestión de economía
Requisitos asociados	• CU-50 Dar/Recibir dinero
Descripción	El sistema almacena el dinero sobre los jugadores, el módulo de economía solo gestiona el intercambio.
Datos específicos	
Comentarios	

Referencia	RI-12
Nombre	Información sobre monstruos
Objetivos asociados	OBJ-12 Gestión de monstruos
Requisitos asociados	• CU-51 Invocar monstruo • CU-52 Crear monstruo • CU-53 Modificar monstruo • CU-54 Eliminar monstruo • CU-55 Crear spawner • CU-56 Modificar spawner • CU-57 Eliminar spawner
Descripción	El sistema deberá almacenar la información correspondiente a los monstruos, en concreto:
Datos específicos	• Nombre de monstruos • Atributos de monstruos • Comportamiento de monstruos • Tipo de ataques de monstruos • Nombre de spawners • Localización de spawners • Tipo de monstruos de spawners
Comentarios	

Referencia	RI-13
Nombre	Información sobre spawners
Objetivos asociados	OBJ-12 Gestión de spawners
Requisitos asociados	• CU-55 Crear spawner • CU-56 Modificar spawner • CU-57 Eliminar spawner
Descripción	El sistema deberá almacenar la información correspondiente a los generadores de monstruos, en concreto:
Datos específicos	• Nombre de spawners • Localización de spawners • Atributos del spawner • Tipo de monstruo en el spawner
Comentarios	

Referencia	RI-14
Nombre	Información sobre configuración global
Objetivos asociados	OBJ-14 Gestión de configuración global
Requisitos asociados	• CU-58 Crear/Modificar configuración global
Descripción	El sistema deberá almacenar la información correspondiente a la configuración global, en concreto:
Datos específicos	• Lugar de aparición de los jugadores • Configuración de daños • Permisos de jugador
Comentarios	

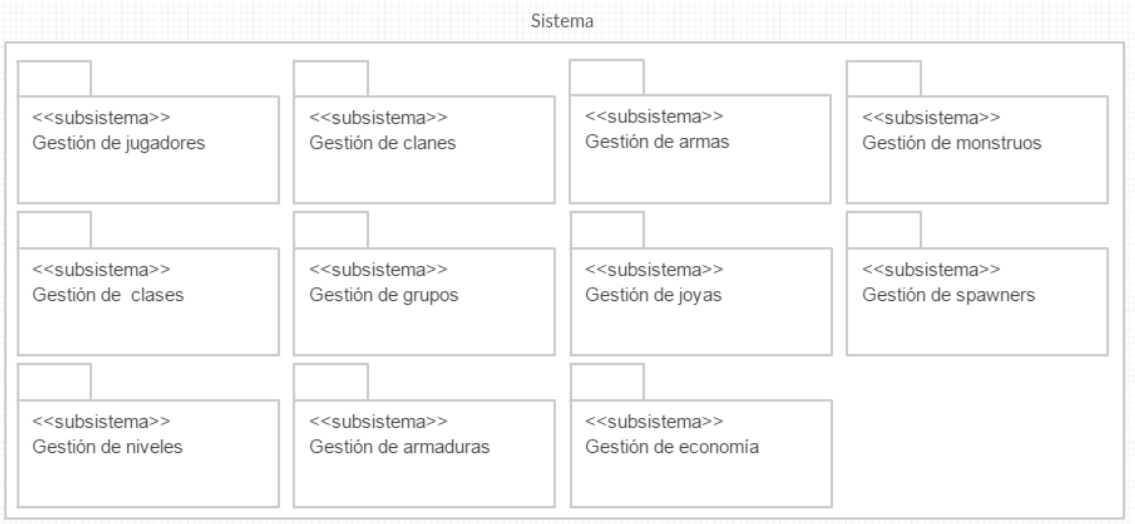
5.1.4 Requisitos funcionales

A continuación se diferenciaran los requisitos de la aplicación de escritorio del complemento, por lo que no siempre se encontrarán los casos de uso en ambas partes. Por ejemplo, las configuraciones generales de los módulos se deberán realizar antes de iniciar el servidor para que surja efecto, mientras que otros son propios de la dinámica del juego.

5.1.4.1 Requisitos del complemento

El complemento modela el videojuego para que obtenga la dinámica que los juegos de rol multijugador traen consigo. El actor principal de este apartado es con diferencia el usuario final o jugador, ya que la tarea del administrador queda subordinada a aspectos de configuración y creación de los elementos que compondrán el videojuego en el siguiente apartado.

En este apartado he incluido el diagrama de subsistemas de diagramas seguido de los diagramas de casos de uso del complemento, separados por subsistemas o módulos que intervienen en esta parte de la solución.

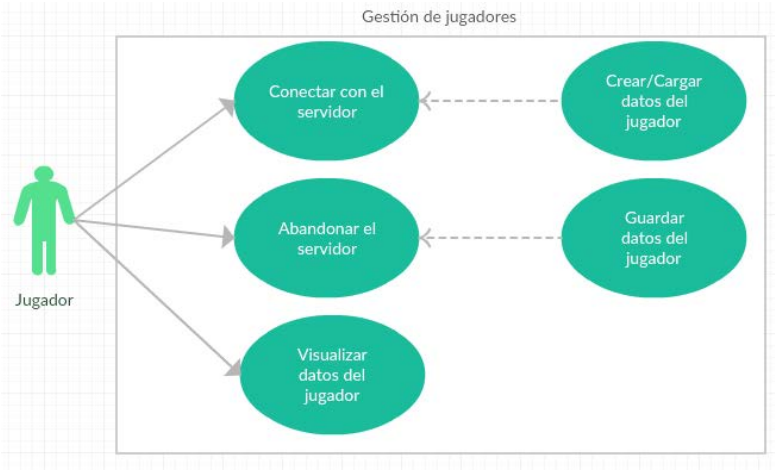


1. Módulo de jugadores

En todo juego MMORPG se manejan infinidad de información, entre esta información la de los personajes de los jugadores, es de entender que al conectar por primera vez al servidor se deberán crear unos datos por defecto y en las próximas cargar los datos almacenados con anterioridad. Al abandonar el servidor se deberá salvar la información actual del personaje del jugador.

La posible información que maneja será los atributos propios de la profesión que seleccione, el clan al que este asociado, el nivel y experiencia en el que se encuentra, entre otras.

La otra función que realiza este módulo es la captura de eventos que serán gestionados por otros módulos, por poner un caso cuando el jugador sube de nivel por un aumento de su experiencia el módulo de clases es el que le aumenta los atributos del personaje.



Referencia	CU-01
Nombre	Conectar con el servidor
Descripción	Ocurre cuando el jugador conecta con el servidor
Actor	Jugador
Relaciones	CU-02 Crear/Cargar datos del jugador
Precondiciones	
Comentarios	

Referencia	CU-02
Nombre	Crear/Cargar datos del jugador
Descripción	Crea las estructuras de datos básica del jugador o las carga en el caso de que ya existieran
Actor	Jugador
Relaciones	
Precondiciones	Para la carga deben existir datos a priori.

Comentarios	

Referencia	CU-03
Nombre	Abandonar el servidor
Descripción	Ocurre cuando el jugador abandona el servidor, ya sea por desconexión propia del jugador o por alguna incidencia de la red.
Actor	Jugador
Relaciones	CU-04 Guardar datos del jugador
Precondiciones	Debe estar conectado al servidor
Comentarios	

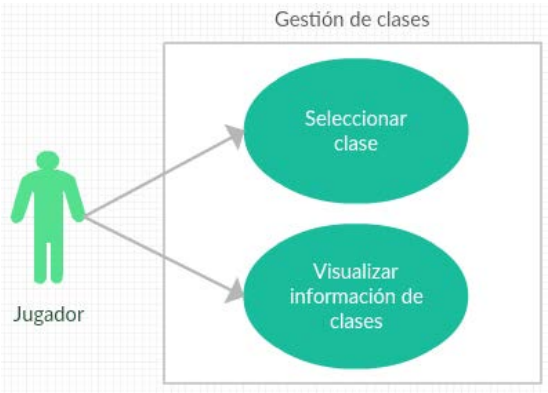
Referencia	CU-04
Nombre	Guardar datos del jugador
Descripción	Guarda los atributos del jugador, incluye atributos de clase, dinero, experiencia, nivel, entre otras.
Actor	Jugador
Relaciones	
Precondiciones	
Comentarios	

Referencia	CU-05
Nombre	Visualizar datos del jugador
Descripción	Detalla los atributos del jugador, incluye atributos de clase, dinero, experiencia, nivel, entre otras.
Actor	Jugador
Relaciones	
Precondiciones	
Comentarios	

2. Módulo de clases

Este módulo cargará las distintas clases con sus atributos desde la configuración al videojuego, permitiendo al jugador seleccionar una clase o profesión.

Este módulo está relacionado directamente con el módulo de jugadores, puesto que los atributos propios de la clase serán aplicados a los datos del jugador.

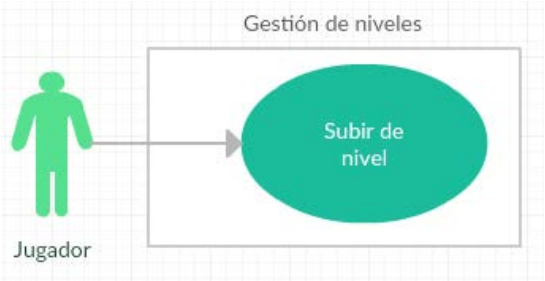


Referencia	CU-06
Nombre	Seleccionar clase
Descripción	El jugador elige una clase o profesión y adquiere los atributos básicos de la clase.
Actor	Jugador
Relaciones	
Precondiciones	
Comentarios	Al elegir una clase el jugador adquiere los atributos propios de la profesión.

Referencia	CU-07
Nombre	Visualizar información de clases
Descripción	El jugador puede visualizar los atributos de las clases creadas en el servidor.
Actor	Jugador
Relaciones	
Precondiciones	
Comentarios	

3. Módulo de niveles

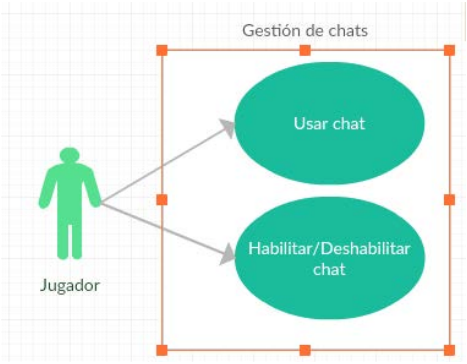
Este módulo se encarga de cargar la configuración de los niveles y de velar por la experiencia de los jugadores. Cuando un cambio de experiencia en un jugador implica subir de nivel al personaje, este módulo se encarga de mejorar los atributos del jugador mediante los datos de su clase almacenados en el módulo de clases.



Referencia	CU-12
Nombre	Subir de nivel
Descripción	El jugador sube de nivel mediante un cambio de su experiencia.
Actor	Jugador
Relaciones	Al subir de nivel el módulo de niveles hace uso del nivel de clases para mejorar los atributos del jugador.
Precondiciones	
Comentarios	

4. Módulo de chat

El módulo de chat se encarga de cargar la configuración del chat y de las interacciones de los usuarios con el chat, sustituyendo el sistema de chat original de Minecraft añadiendo canales. El módulo de chat se encarga de capturar los eventos asociados al envío de mensajes de los jugadores en el servidor.

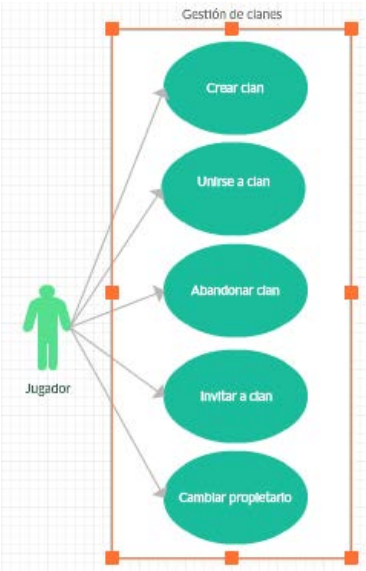


Referencia	CU-15
Nombre	Usar el chat
Descripción	El jugador envía mensajes a través del chat.
Actor	Jugador
Relaciones	
Precondiciones	
Comentarios	

Referencia	CU-16
Nombre	Habilitar/Deshabilitar chats
Descripción	El jugador puede habilitar o deshabilitar chats en el servidor con tal de dejar de escuchar dichos canales.
Actor	Jugador
Relaciones	Este caso de uso tiene dos actores realmente pero en el complemento su actor es el jugador, en la aplicación de escritorio el actor es el administrador quien se encarga de gestionarlo habilitando o deshabilitando los canales.
Precondiciones	
Comentarios	

5. Módulo de clanes

El módulo de clanes se encarga de manejar los clanes y la configuración existente en el servidor, gestiona los miembros de los clanes y ofrece herramientas a los jugadores para su interacción.



Referencia	CU-18
Nombre	Crear clan
Descripción	El jugador crea un clan nuevo.
Actor	Jugador
Relaciones	
Precondiciones	No debe pertenecer a ningún clan y cumplir con los requisitos necesarios.
Comentarios	

Referencia	CU-19
Nombre	Unirse a clan
Descripción	El jugador acepta una invitación de un clan para unirse.
Actor	Jugador
Relaciones	CU-39 Invitar a clan
Precondiciones	Debe existir una invitación del jugador al clan para aceptarla.
Comentarios	

Referencia	CU-20
Nombre	Abandonar el clan
Descripción	El jugador abandona el clan en el que se encuentra.
Actor	Jugador
Relaciones	
Precondiciones	Debe pertenecer a un clan.
Comentarios	

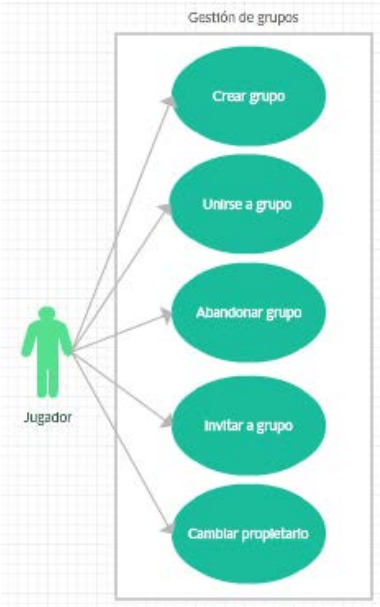
Referencia	CU-21
Nombre	Invitar a clan
Descripción	El jugador envía una invitación de unión al clan a otro jugador.
Actor	Jugador
Relaciones	
Precondiciones	El jugador que invita al clan debe pertenecer a uno y el otro jugador no debe pertenecer a un clan.
Comentarios	

Referencia	CU-22
Nombre	Cambiar propietario del clan
Descripción	El jugador propietario del clan pasa la propiedad del clan a otro jugador.
Actor	Jugador
Relaciones	
Precondiciones	Ambos jugadores deben pertenecer al mismo clan. El jugador que realiza la acción debe ser el propietario del clan.

Comentarios	

6. Módulo de grupos

El módulo de grupos se encarga de gestionar la configuración y los grupos que se creen en el servidor. La diferencia principal entre un clan y un grupo es que la información de los grupos es volátil y por tanto no se almacena realmente información cuando se producen cambios. Al igual que el módulo de clanes ofrecerá herramientas al jugador para interactuar con los grupos.



Referencia	CU-25
Nombre	Crear grupo
Descripción	El jugador crea un nuevo grupo.
Actor	Jugador
Relaciones	
Precondiciones	El jugador no debe pertenecer a un grupo.
Comentarios	

Referencia	CU-26
Nombre	Unirse a grupo
Descripción	El jugador acepta una invitación al grupo.
Actor	Jugador
Relaciones	CU-41 Invitar a grupo
Precondiciones	El jugador no debe pertenecer a un grupo y recibir una invitación.
Comentarios	

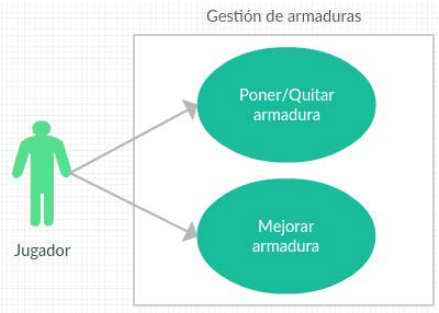
Referencia	CU-27
Nombre	Abandonar grupo
Descripción	El jugador abandona el grupo.
Actor	Jugador
Relaciones	
Precondiciones	El jugador debe pertenecer a un grupo
Comentarios	

Referencia	CU-28
Nombre	Invitar a grupo
Descripción	El jugador invita a otro jugador a unirse al grupo.
Actor	Jugador
Relaciones	
Precondiciones	El jugador debe pertenecer a un grupo.
Comentarios	

Referencia	CU-29
Nombre	Cambiar propietario del grupo
Descripción	El jugador propietario del grupo pasa la propiedad del grupo a otro jugador.
Actor	Jugador
Relaciones	
Precondiciones	Ambos jugadores deben pertenecer al mismo grupo. El jugador que realiza la acción debe ser el propietario del grupo.
Comentarios	

7. Módulo de armaduras

El módulo de armaduras gestiona la carga de armaduras en el servidor. El jugador interactúa con las partes de la armadura equipándolas y quitándolas del equipo, modificando así sus atributos.

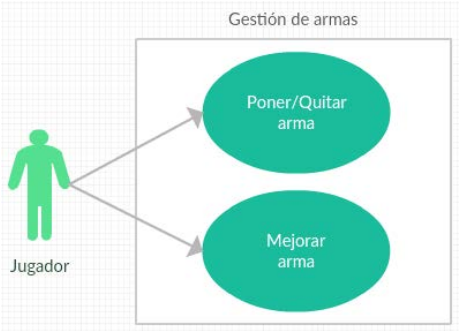


Referencia	CU-32
Nombre	Poner/Quitar armadura
Descripción	El jugador equipa o quita una parte de armadura del equipo.
Actor	Jugador
Relaciones	
Precondiciones	Deben existir armaduras a priori.
Comentarios	

Referencia	CU-33
Nombre	Mejorar armadura
Descripción	El jugador mejora una parte de armadura.
Actor	Jugador
Relaciones	
Precondiciones	Deben existir armaduras mejorables a priori.
Comentarios	

8. Módulo de armas

El módulo de armas gestiona la carga de armas en el servidor. El jugador interactúa con las armas equipándolos y quitándolos del equipo, modificando así sus atributos.



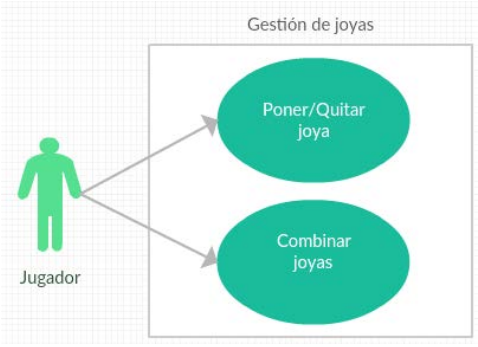
Referencia	CU-38
Nombre	Poner/Quitar arma

Descripción	El jugador equipa o quita un arma del equipo.
Actor	Jugador
Relaciones	
Precondiciones	Deben existir armas a priori.
Comentarios	

Referencia	CU-39
Nombre	Mejorar arma
Descripción	El jugador mejora un arma.
Actor	Jugador
Relaciones	
Precondiciones	Deben existir armas mejorables a priori.
Comentarios	

9. Módulo de joyas

El módulo de joyas gestiona la carga de joyas en el servidor. El jugador interactúa con las joyas equipándolos y quitándolos del equipo, modificando así sus atributos.



Referencia	CU-44
Nombre	Poner/Quitar joyas en el equipo
Descripción	El jugador equipa o quita una joya del equipo.
Actor	Jugador
Relaciones	
Precondiciones	Deben existir joyas a priori.
Comentarios	

Referencia	CU-45
Nombre	Combinar joyas
Descripción	El jugador combina dos joyas

Actor	Jugador
Relaciones	
Precondiciones	Deben existir joyas combinables entre ellas a priori.
Comentarios	

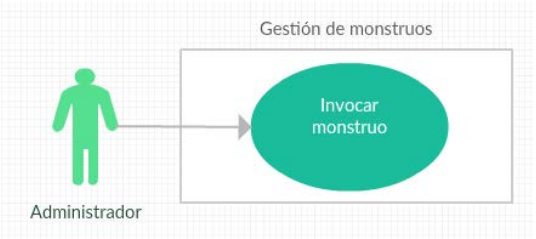
10. Módulo de economía

El módulo de economía se encarga de los procesos de pagos entre jugadores.

Referencia	CU-50
Nombre	Dar/Recibir dinero
Descripción	El jugador da/recibe dinero a/de otro jugador.
Actor	Jugador
Relaciones	
Precondiciones	El jugador que da debe tener dinero.
Comentarios	

11. Módulo de monstruos

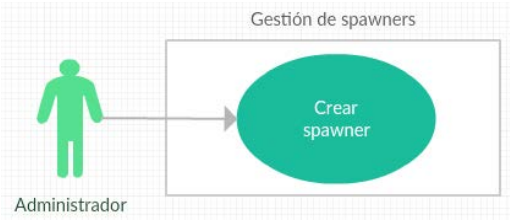
El módulo de monstruos se encarga de la carga de la configuración sobre monstruos. Permite al administrador invocar monstruos.



Referencia	CU-51
Nombre	Invocar monstruo
Descripción	El administrador invoca en su posición un monstruo.
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un monstruo a priori para su invocación.
Comentarios	

12. Módulo de spawners

Este módulo se encarga de cargar la información de los generadores de monstruos, activándolos y desactivándolos si son necesarios. El administrador puede crear nuevos generadores de monstruos.



Referencia	CU-55
Nombre	Crear spawner
Descripción	El administrador crea un nuevo spawner o generador de monstruos.
Actor	Administrador
Relaciones	
Precondiciones	Al menos debe existir un monstruo para asociar al generador de monstruos.
Comentarios	

5.1.4.2 Requisitos de la aplicación

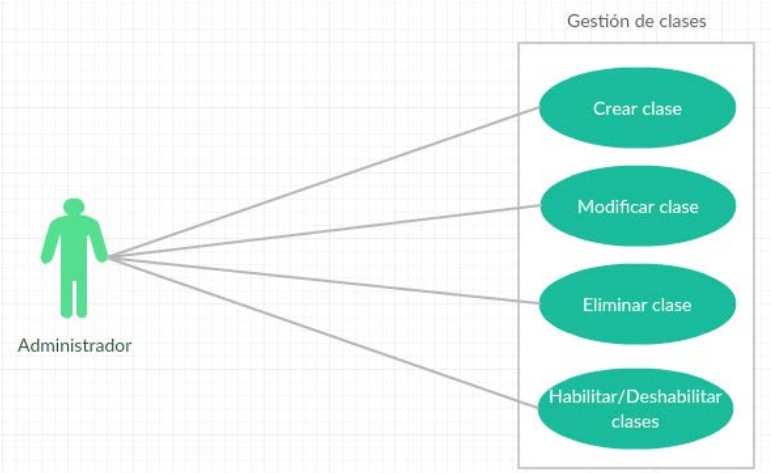
La de escritorio tendrá como objetivo crear configuraciones para el propio complemento, por ello será necesario que ofrezca la creación, modificación y eliminación de las configuraciones de una manera modular, independientes entre ellas en la medida de lo posible, de manera que sean fácilmente exportables al servidor y reemplazadas en cualquier momento. El actor principal que realiza las configuraciones es el propio administrador del servidor.

En este apartado he incluido el diagrama de subsistemas de diagramas seguido de los diagramas de casos de uso de la aplicación de escritorio, separados por subsistemas o módulos que intervienen en esta parte de la solución.

A continuación se detallarán las funcionalidades requeridas en casos de uso con prototipos de las interfaces.

1. Módulo de clases

En todo juego de rol clásico se encuentran distintas clases aplicables a nuestros personajes. Cuando se crea un personaje, este debe elegir una clase que le será asignada para siempre y que le supeditará a cierto tipo de combate. Este módulo se encargará de gestionar las clases del juego.



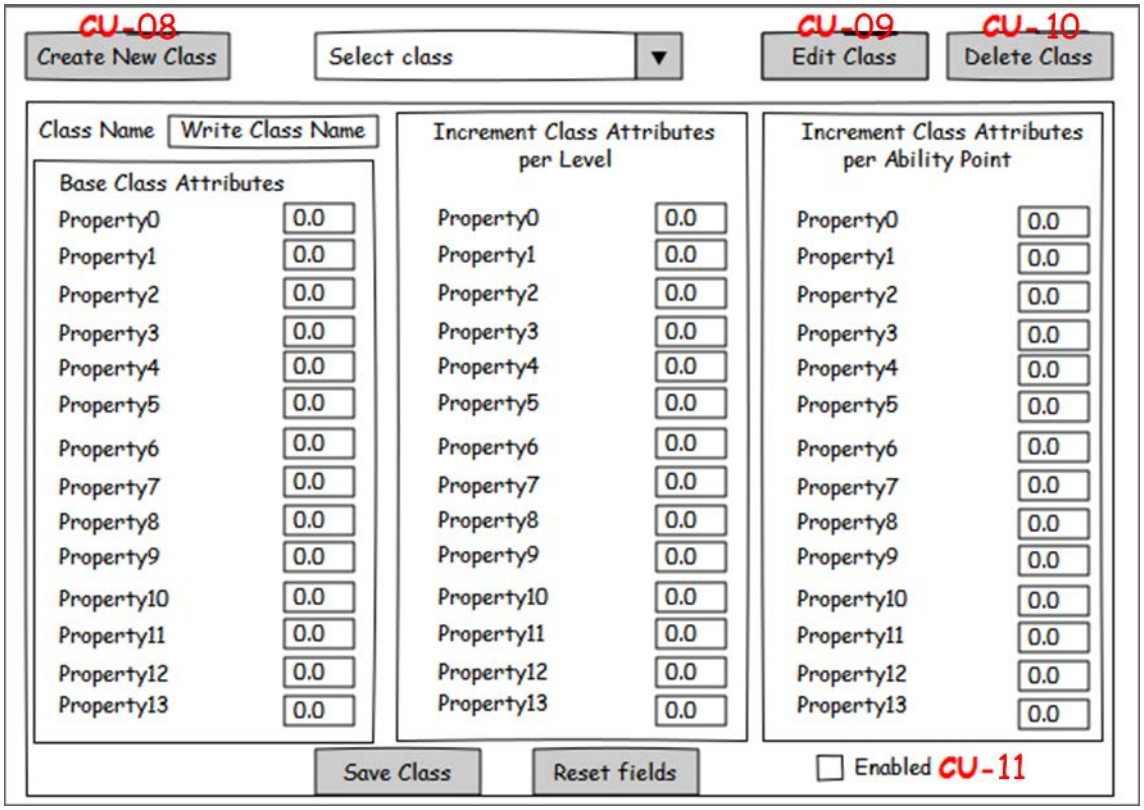
Referencia	CU-08
Nombre	Crear clase
Descripción	Crea clase de jugador
Actor	Administrador
Relaciones	
Precondiciones	
Comentarios	

Referencia	CU-09
Nombre	Modificar clase
Descripción	Modifica una clase de jugador
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una clase para modificar
Comentarios	

Referencia	CU-10
Nombre	Eliminar clase
Descripción	Elimina una clase de jugador
Actor	Administrador
Relaciones	
Precondiciones	Debe existir la clase de jugador
Comentarios	

Referencia	CU-11

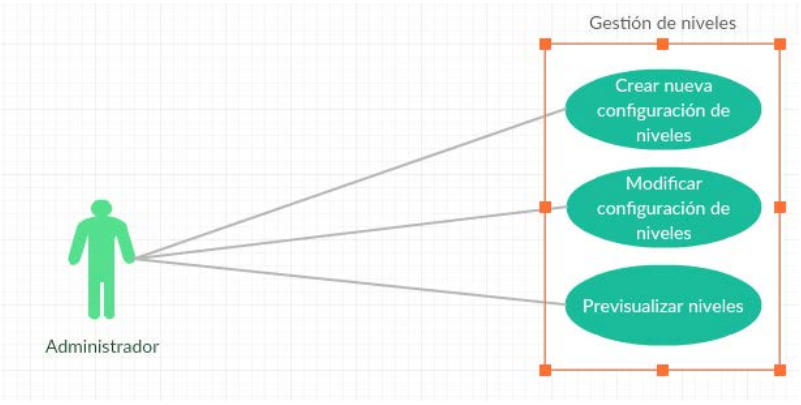
Nombre	Habilitar/Deshabilitar clase
Descripción	Habilita o Deshabilita una clase de jugador
Actor	Administrador
Relaciones	
Precondiciones	Debe existir la clase de jugador
Comentarios	



2. Módulo niveles

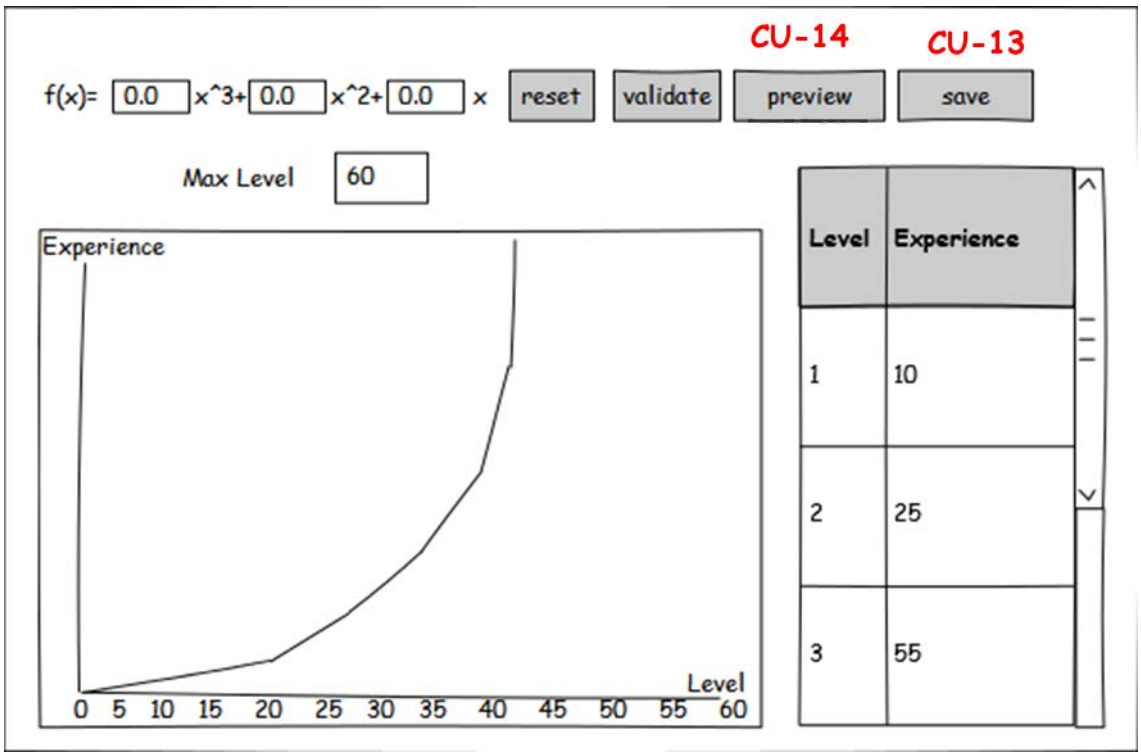
Los sistemas de niveles son una característica heredada de los juegos RPG[20] clásicos, donde se dividen las características básicas de un jugador y dependiendo de su clase tienen un puntaje. Este sistema hace que cada experiencia dentro del juego sea única e individualizada para cada jugador de una misma clase, enriqueciendo la experiencia de los jugadores. Este módulo gestionará la configuración de niveles.

[20] RPG: Role-Playing Game, en español juegos de rol.



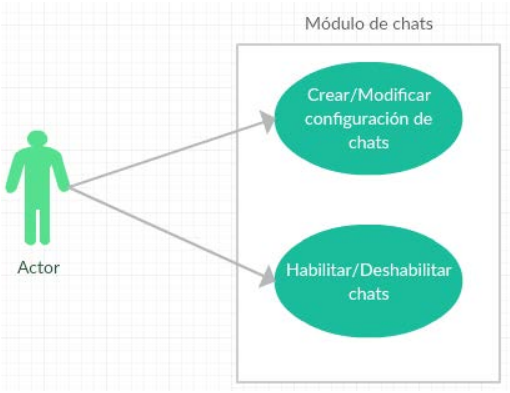
Referencia	CU-13
Nombre	Crear/Modificar configuración de niveles
Descripción	Crear/Modificar configuración de niveles
Actor	Administrador
Relaciones	
Precondiciones	
Comentarios	Por defecto siempre existe una configuración de niveles, en caso de que hubiera sido modificada se mostrará la configuración actual

Referencia	CU-14
Nombre	Visualización de configuración de niveles
Descripción	Visualiza la configuración de niveles en un gráfico.
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración ya creada y validada por la aplicación
Comentarios	Al pre visualizar se mostrará una lista nivel-experiencia de todos los niveles acompañado de una gráfica que mostrará la pendiente de los niveles en su incremento ascendente.



3. Módulo de chat

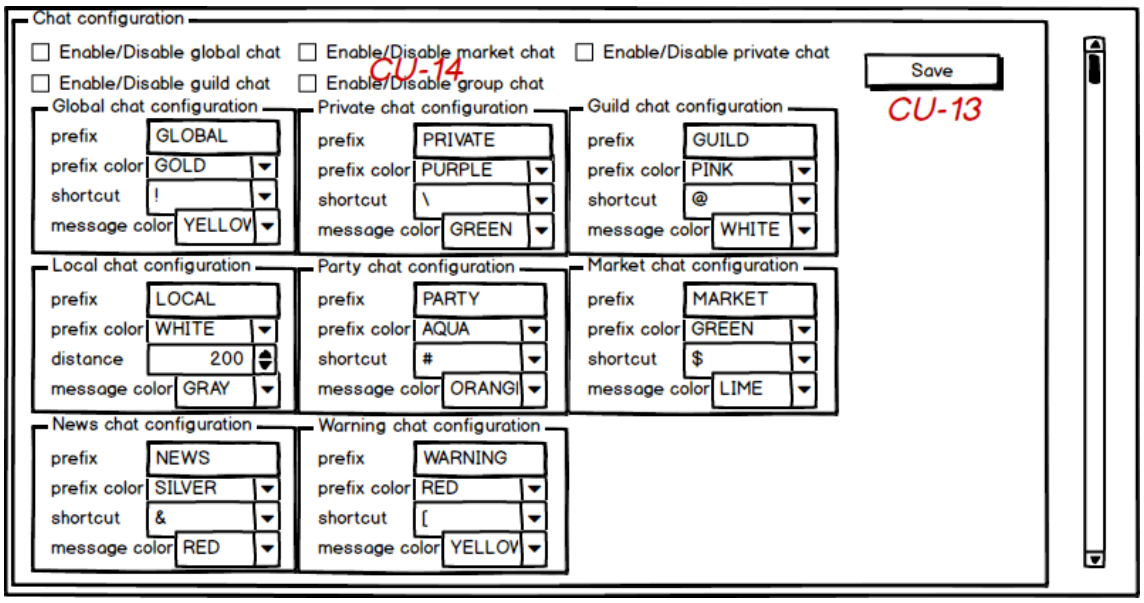
El módulo de chat se encarga de gestionar los distintos canales de comunicación de los jugadores y administradores, mediante la configuración de prefijos, colores y habilitando o deshabilitando canales.



Referencia	CU-13
Nombre	Crear/Modificar configuración de chats
Descripción	Crea o modifica la configuración de chats, modificando colores y prefijos de los distintos canales, distancia de escucha, entre otras opciones.
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración a priori.

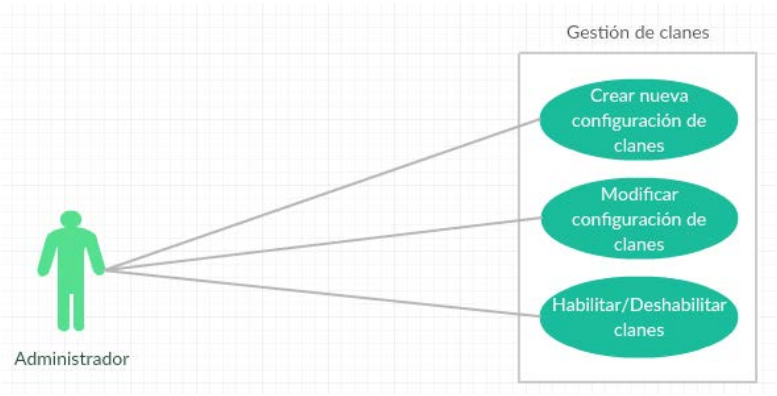
Comentarios	La distancia de escucha es una propiedad de los chats de carácter local, en los cuáles los receptores deben estar a menos de cierta distancia para escuchar al emisor.

Referencia	CU-14
Nombre	Habilitar/Deshabilitar chats
Descripción	Habilita o deshabilita los chats que sean necesarios.
Actor	Administrador
Relaciones	
Precondiciones	
Comentarios	



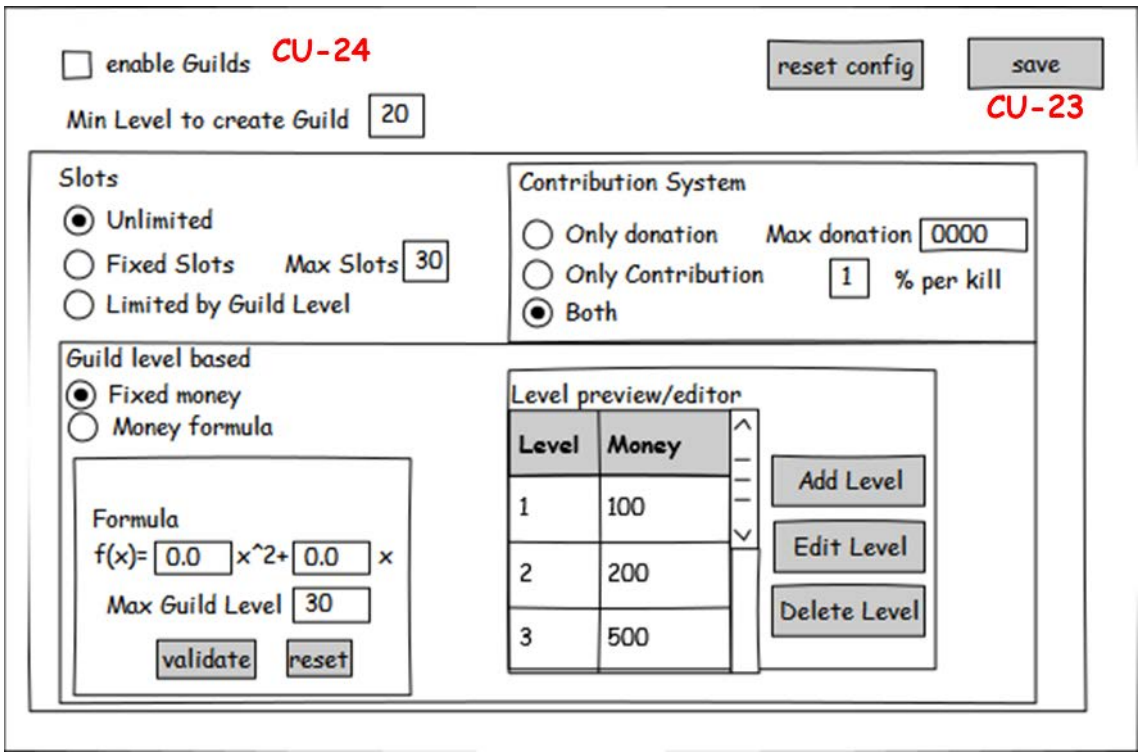
4. Módulo de clanes

Los clanes son agrupaciones de jugadores unidos en una pequeña comunidad de la que todos son participes. Este módulo se encarga de gestionar la configuración sobre este tipo de agrupaciones.



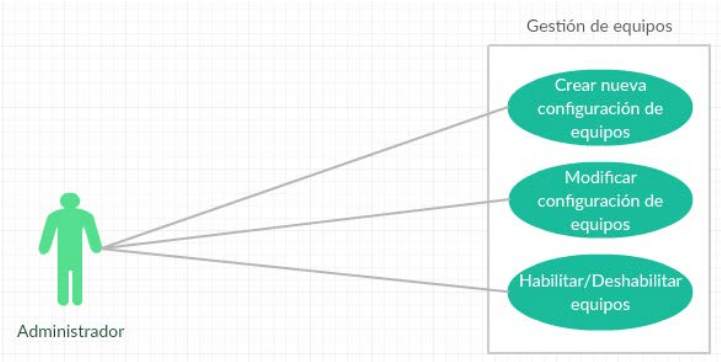
Referencia	CU-23
Nombre	Crear/Modificar configuración de clanes
Descripción	Crea o modifica la configuración actual de clanes
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración a priori para su modificación
Comentarios	Por defecto se crea una configuración de clanes

Referencia	CU-24
Nombre	Habilitar/Deshabilitar clanes
Descripción	Habilita o deshabilita los clanes en el juego
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración a priori para su modificación
Comentarios	Por defecto se crea una configuración de clanes



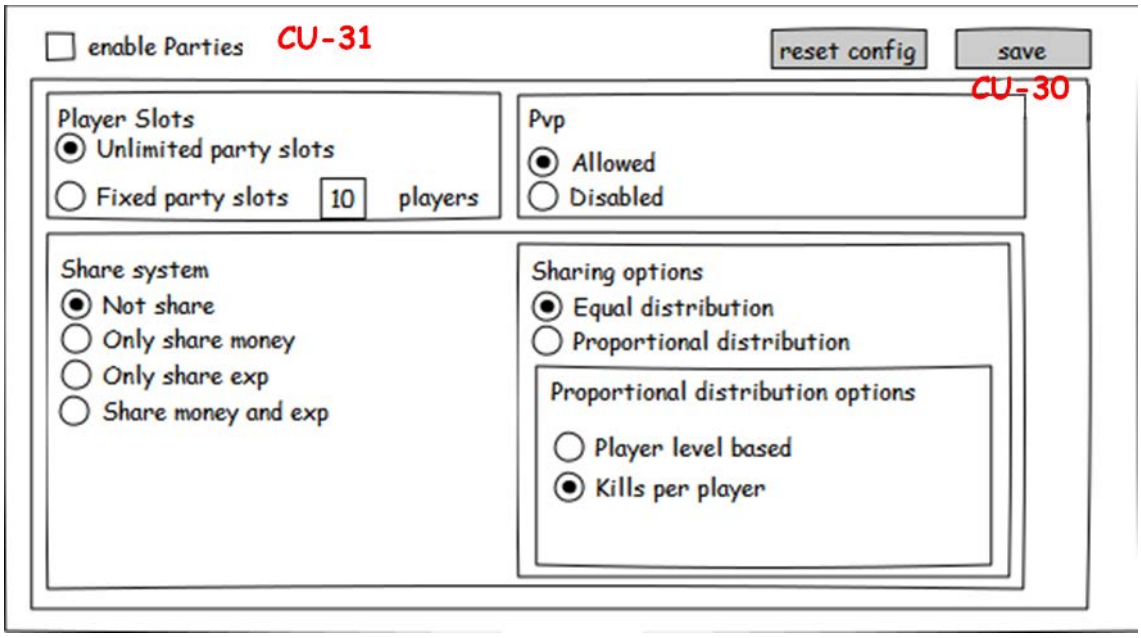
5. Módulo de grupos

Los grupos, llamados *parties en inglés*, son agrupaciones de jugadores que a diferencia de los clanes son de duración determinada, en la que los jugadores comparten los beneficios de la batalla (experiencia, objetos dejados caer por las entidades o dinero). Este módulo se encargará de la configuración de los grupos.



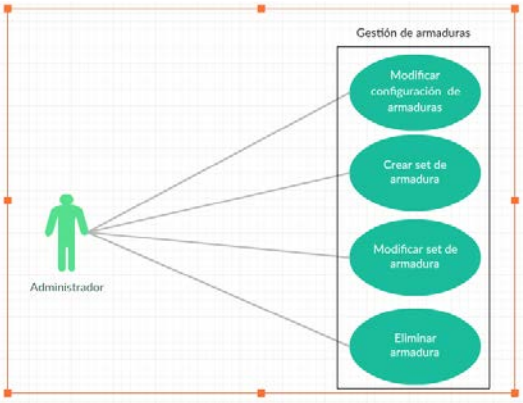
Referencia	CU-30
Nombre	Crear/Modificar configuración de grupos
Descripción	Crea o modifica la configuración de los grupos en el juego
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración a priori para su modificación
Comentarios	Por defecto se crea una configuración de grupos

Referencia	CU-31
Nombre	Habilitar/Deshabilitar los grupos
Descripción	Habilita o Deshabilita los grupos en el juego
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración a priori para su modificación
Comentarios	Por defecto se crea una configuración de grupos



6. Módulo de armaduras

Las armaduras son equipos de nuestros jugadores, que aportan mayor defensa ante ataques, además pueden ofrecer la posibilidad de añadir características especiales a las partes de la armadura, haciendo así que cada parte sea única. Este módulo permite la gestión equipos de armadura personalizados.



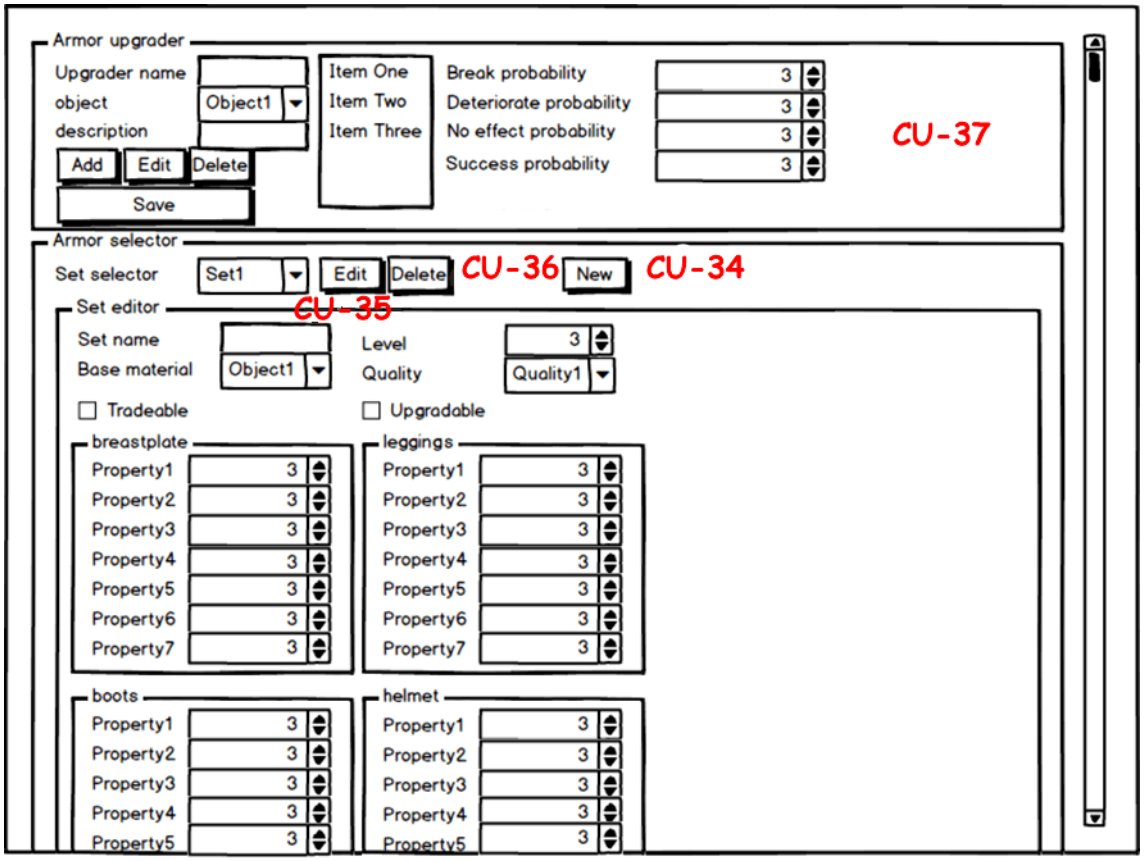
Referencia	CU-37
Nombre	Crear/Modificar configuración general de armaduras
Descripción	Modifica el objeto que mejora las armas
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración a priori para su modificación
Comentarios	Por defecto se crea una configuración de armaduras

Referencia	CU-34
Nombre	Crear un set de armadura
Descripción	Crea un set compuesto de cuatro partes de una armadura
Actor	Administrador
Relaciones	
Precondiciones	
Comentarios	

Referencia	CU-35
Nombre	Modificar set de armadura
Descripción	Modifica el set compuesto de cuatro partes de una armadura
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un set a priori para su modificación
Comentarios	

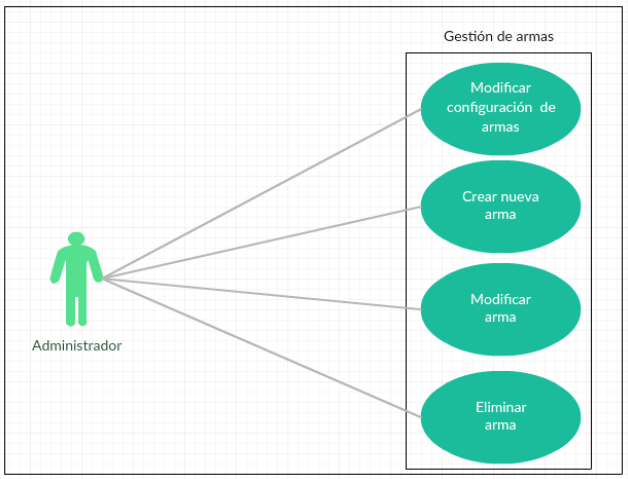
Referencia	CU-36

Nombre	Elimina un set de armadura
Descripción	Elimina un set compuesto de cuatro partes de una armadura
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un set a priori para su eliminación
Comentarios	Al eliminar deja de existir en la configuración, pero en el juego pueden seguir existiendo



7. Módulo de armas

Las armas son parte del equipo de los jugadores, pueden o no ser de un tipo específico de clase y otorgar mayor poder al jugador. Este módulo permite la gestión de armas personalizadas que luego serán usados por el complemento en el juego.



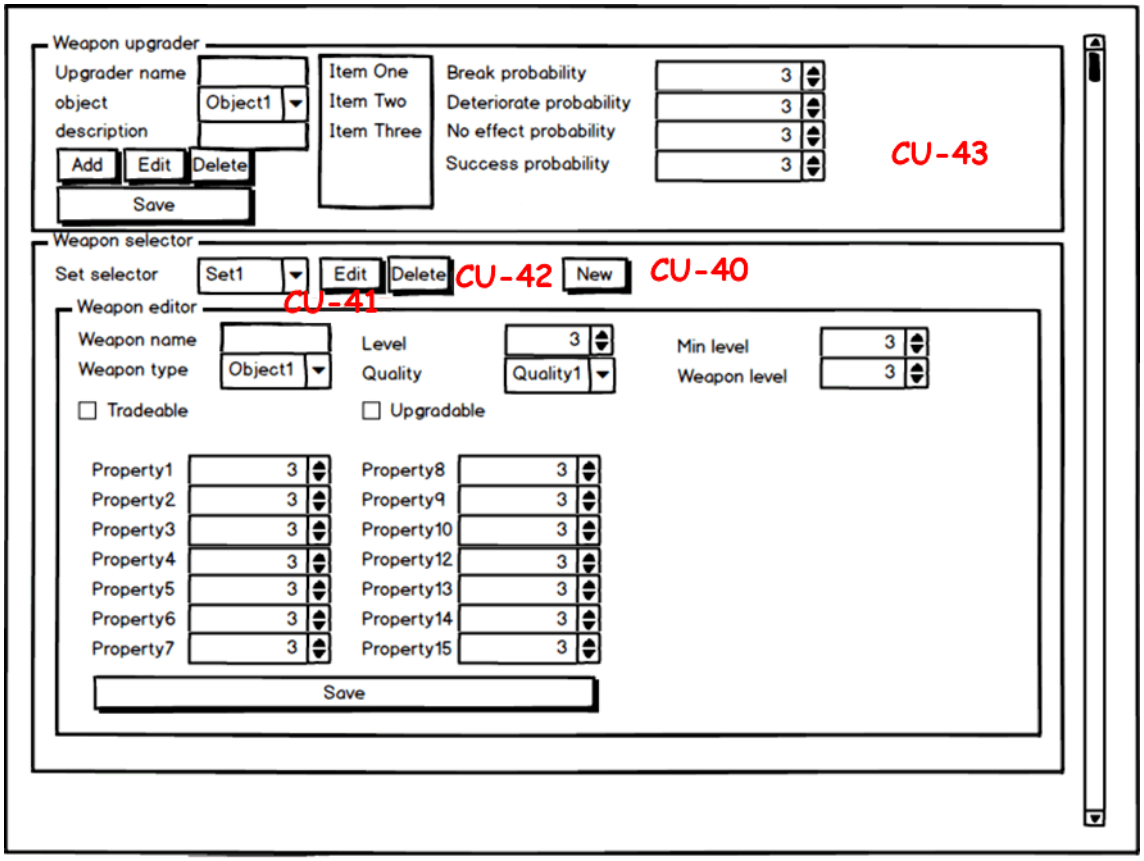
Referencia	CU-43
Nombre	Modificar configuración de armas
Descripción	Modifica el objeto mejorador de armas
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración a priori para su modificación
Comentarios	Por defecto se crea una configuración de armas

Referencia	CU-40
Nombre	Crear una nueva arma
Descripción	Crea un arma
Actor	Administrador
Relaciones	
Precondiciones	
Comentarios	

Referencia	CU-41
Nombre	Modificar un arma
Descripción	Modifica el arma
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una arma a priori para su modificación
Comentarios	

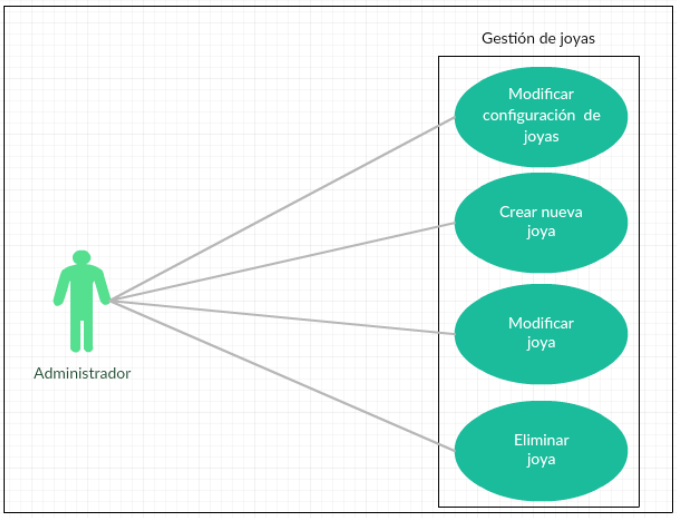
Referencia	CU-42

Nombre	Eliminar arma
Descripción	Elimina el arma seleccionada
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un arma a priori para su eliminación
Comentarios	Al eliminar deja de existir en la configuración, pero en el juego pueden seguir existiendo



8. Módulo de joyas

Las joyas son objetos con efectos especiales sobre los atributos del jugador, pueden ser obtenidos de muchas maneras. Este módulo permite la gestión de joyas personalizadas.



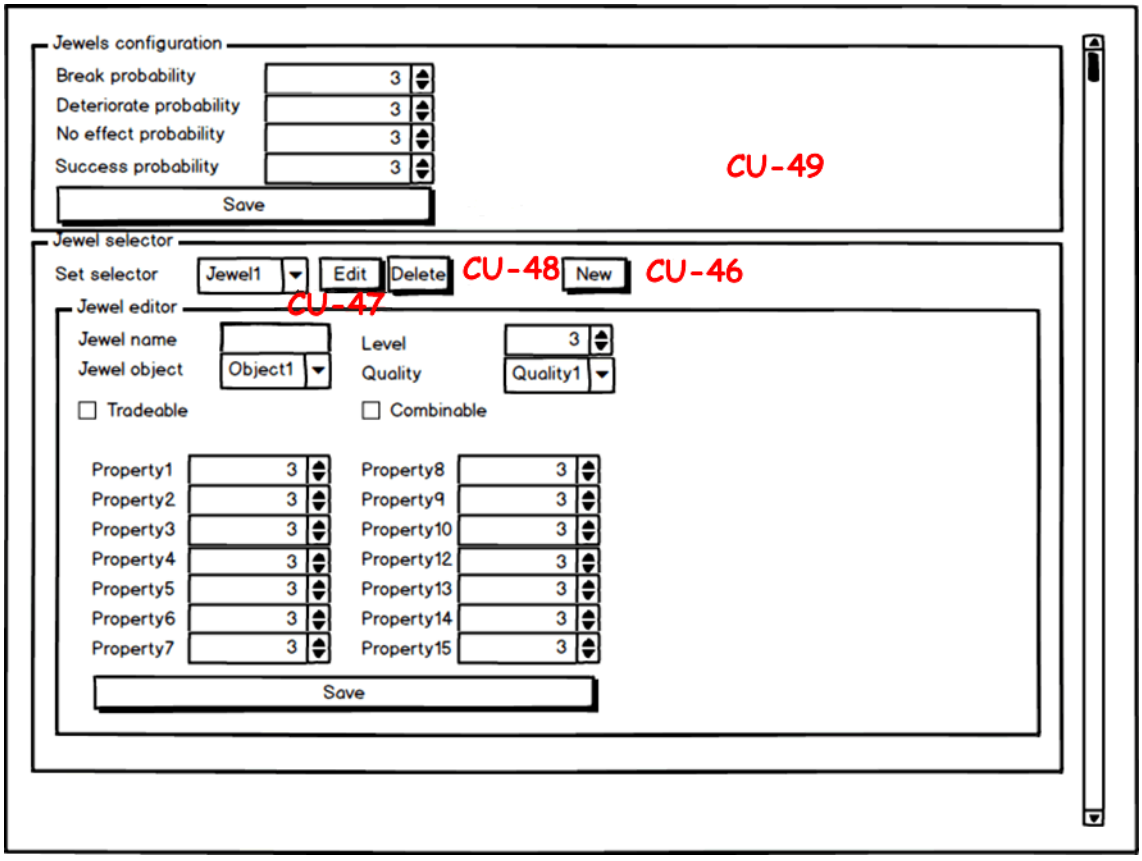
Referencia	CU-49
Nombre	Crear/Modificar configuración de joyas
Descripción	Modifica las probabilidades de mejora de joyas
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una configuración a priori para su modificación
Comentarios	Por defecto se crea una configuración de joyas

Referencia	CU-46
Nombre	Crear nueva joya
Descripción	Crea una joya
Actor	Administrador
Relaciones	
Precondiciones	
Comentarios	

Referencia	CU-47
Nombre	Modificar joya
Descripción	Modifica una joya seleccionada
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una joya a priori para su modificación
Comentarios	

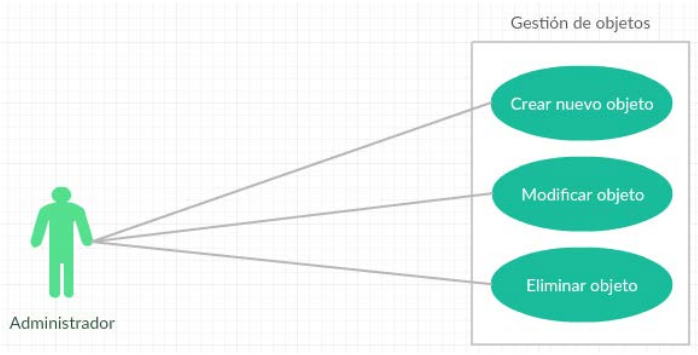
Referencia	CU-48

Nombre	Eliminar joya
Descripción	Elimina una joya seleccionada
Actor	Administrador
Relaciones	
Precondiciones	Debe existir una joya a priori para su eliminación
Comentarios	Al eliminar deja de existir en la configuración, pero en el juego pueden seguir existiendo



9. Módulo de objetos

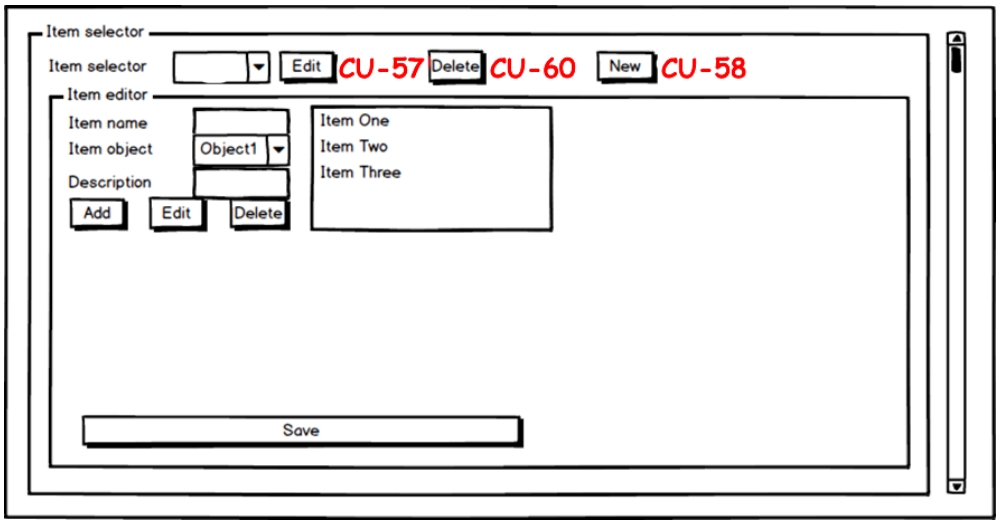
Los objetos con los que interactuará el jugador con variopintos fines (misiones, comercio) y que pueden ser obtenidos de muchas maneras. Este módulo permite la gestión de objetos personalizados.



Referencia	CU-58
Nombre	Crear un nuevo objeto
Descripción	Crea un objeto
Actor	Administrador
Relaciones	
Precondiciones	
Comentarios	

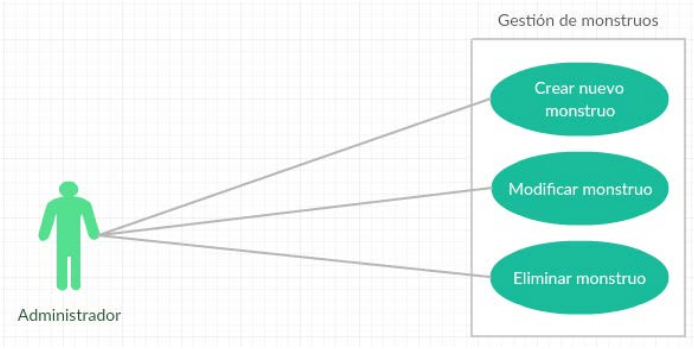
Referencia	CU-29
Nombre	Modificar un objeto
Descripción	Modifica un objeto seleccionado
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un objeto a priori para su modificación
Comentarios	

Referencia	CU-60
Nombre	Eliminar un objeto
Descripción	Elimina un objeto seleccionado
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un objeto a priori para su eliminación
Comentarios	Al eliminar deja de existir en la configuración, pero en el juego pueden seguir existiendo



10. Módulo de monstruos

Los monstruos son entidades que se podrán generar mediante el complemento, serán altamente configurables debido a la configuración de los atributos y del comportamiento del monstruo. Este módulo permitirá la gestión de monstruos personalizados.

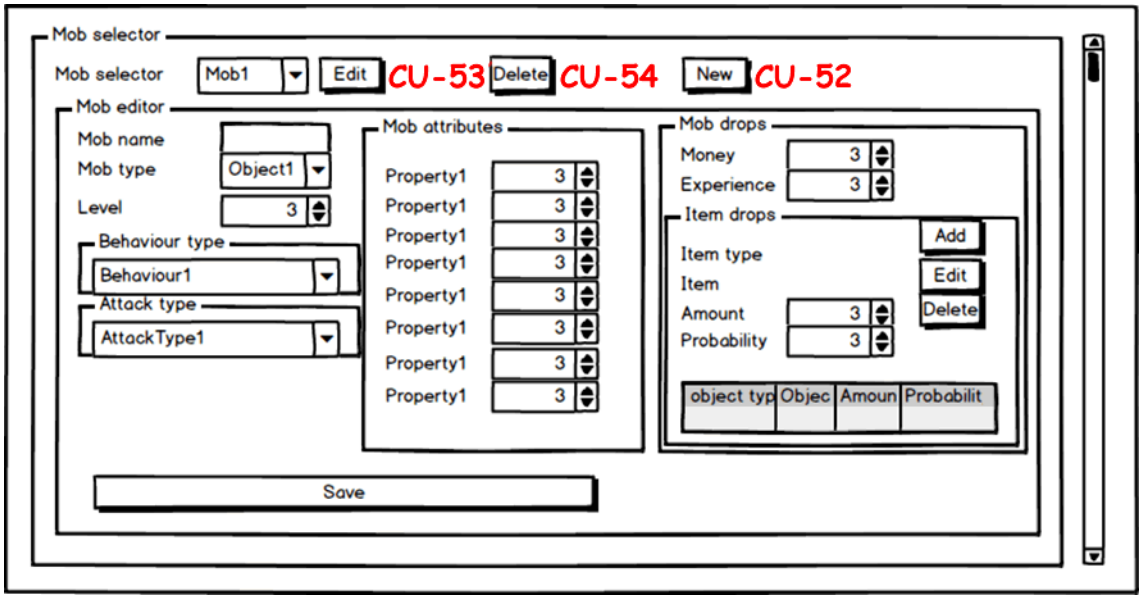


Referencia	CU-52
Nombre	Crear nuevo monstruo
Descripción	Crea un nuevo monstruo
Actor	Administrador
Relaciones	
Precondiciones	
Comentarios	

Referencia	CU-53
Nombre	Modificar monstruo
Descripción	Modifica el monstruo seleccionado
Actor	Administrador
Relaciones	

Precondiciones	Debe existir un monstruo a priori para su modificación
Comentarios	

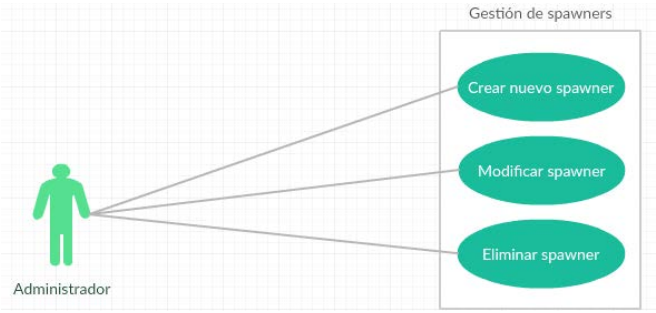
Referencia	CU-54
Nombre	Eliminar monstruo
Descripción	Eliminar el monstruo seleccionado
Actor	Administrador
Relaciones	Módulo de spawners
Precondiciones	Debe existir un monstruo a priori para su modificación
Comentarios	Al tener una relación directa con el módulo de spawners, al eliminar un monstruo asociado a un spawner, el spawner será eliminado



11. Módulo de generadores de monstruos

Los *spawners*[21]son generadores de entidades que se disponen en un punto exacto del mapa para generar las entidades asociadas. Este módulo permitirá la gestión de *spawners.*

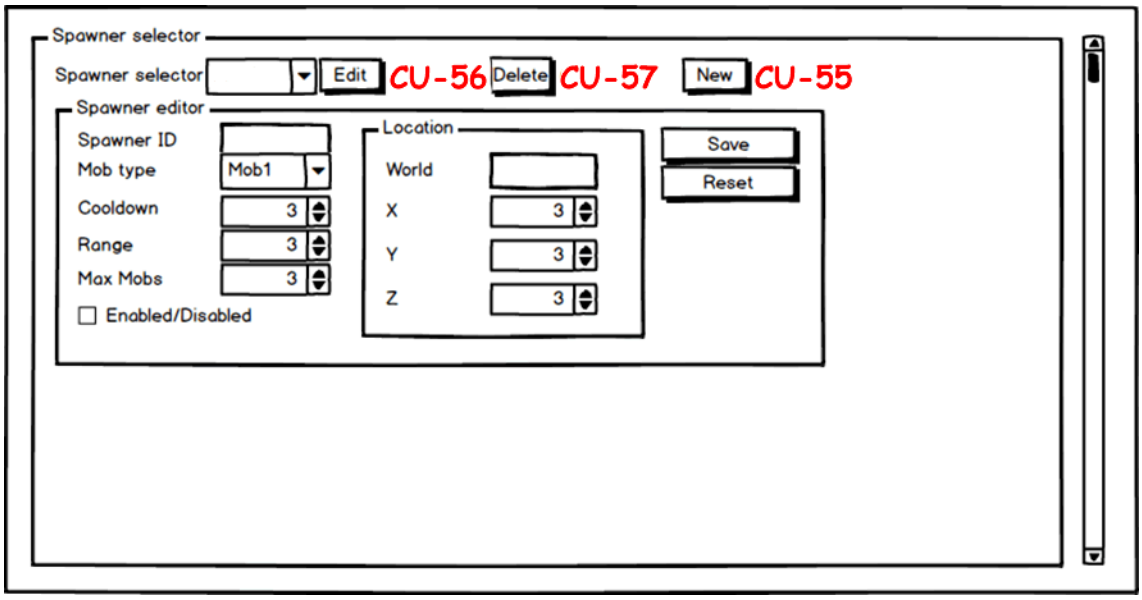
[21] *Spawner*: desovadero en español, es un generador de entidades, en este caso monstruos.



Referencia	CU-55
Nombre	Crear nuevo spawner
Descripción	Crea un spawner
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un monstruo a priori para su creación
Comentarios	

Referencia	CU-56
Nombre	Modificar spawner
Descripción	Modifica un spawner seleccionado
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un spawner a priori para su modificación
Comentarios	

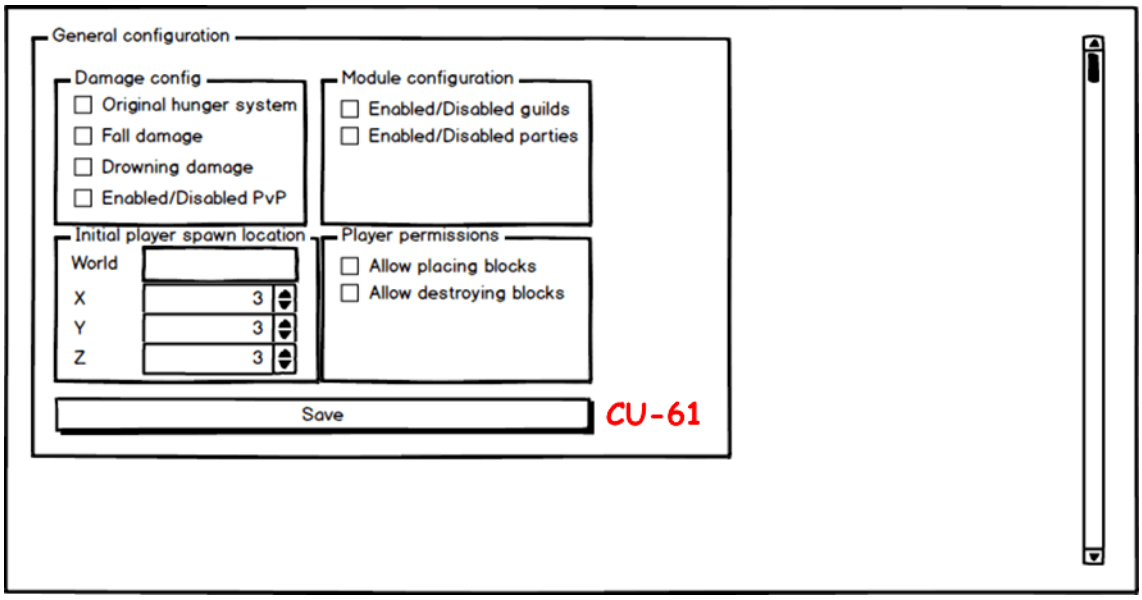
Referencia	CU-57
Nombre	Eliminar spawner
Descripción	Elimina un spawner
Actor	Administrador
Relaciones	
Precondiciones	Debe existir un spawner a priori para su eliminación
Comentarios	



12. Módulo de configuración general

En el módulo de configuración general se pueden configurar aspectos como el hambre, daños por caída o ahogamiento, habilitar o deshabilitar módulos completos y aspectos generales del complemento.

Referencia	CU-61
Nombre	Crear/Modificar configuración general
Descripción	Modifica la configuración general del complemento
Actor	Administrador
Relaciones	Módulos clanes y grupos
Precondiciones	Debe existir una configuración a priori para su modificación
Comentarios	



5.2 Análisis de las soluciones

La solución al problema que plantea este TFG no es único, por lo que se analizarán individualmente los pros y los contras, para finalmente y tras establecer un criterio de elección, decidir la que más se adecue al proyecto.

A continuación en sub-apartados se analizarán las posibles soluciones.

Solución desarrollada mediante complemento

Una primera solución es desarrollar un complemento o *plugin* para un servidor Minecraft que modele los requisitos citados anteriormente.

Una ventaja que ofrecen los complementos en servidores Minecraft es que no es necesario adaptar el cliente al servidor, el jugador podrá conectarse y jugar con normalidad.

Otra ventaja es que la configuración del complemento se realiza separada de la del servidor, mejorando la visibilidad global del complemento, mediante una interfaz preparada exclusivamente para su fin, la de configurar un complemento MMORPG.

Otra ventaja es que a la hora de actualizar un complemento de un servidor para añadir nuevas funcionalidades, el jugador no tiene por qué realizar ninguna actualización sobre la parte cliente.

En contrapartida, la API de Minecraft no está completa, no permite modificar los atributos de los monstruos que trae por defecto al gusto como un servidor quisiera, y las primeras soluciones a este problema que se encontraron dejaron mucho que desear, pues trataban de capturar cada evento y modificar el comportamiento del monstruo en ejecución, lo cual es un malgasto de recursos de computación y memoria. Salvando un poco el obstáculo y mediante ingeniería inversa se pudo discernir una posible solución tangible a este problema, reescribir las clases de los monstruos en Minecraft permitiendo así modificar sus atributos, comportamiento y registrarlos en el servidor como propios del mismo, creando además métodos para que un mismo monstruo se comporte de distintas maneras. Un inconveniente de este último método es tener que tratar de codificar las 32 clases de monstruos que existen en Minecraft, reescribiendo todo su código clase por clase, con código técnicamente ofuscado, que se debe haber de-compilado con anterioridad e investigado mediante ingeniería inversa que funciones tienen sus partes de código ofuscado.

Solución desarrollada mediante modificación del servidor-cliente

La segunda solución es desarrollar una modificación, o *mod* en inglés, tanto del servidor de Minecraft como de la parte cliente que implemente los requisitos citados anteriormente.

Una ventaja que ofrece una modificación de este tipo es que son desarrolladas exclusivamente para su fin, en este caso un MMORPG debe funcionar de manera óptima. Existen en la actualidad este tipo de modificaciones sobre Minecraft, que requieren de descargas adicionales de las modificaciones sobre los lanzadores del juego para que el jugador pueda ingresar a los servidores modificados.

A pesar de ser una solución eficiente, un gran inconveniente es tener que modificar gran parte de código del servidor y del cliente para crear estas modificaciones, teniendo en cuenta que el código obtenido mediante de-compilación estará técnicamente ofuscado, en gran parte, al igual que en la solución anterior.

Otro inconveniente es que se limita el acceso y se obliga a jugadores a tener instaladas estas modificaciones del cliente de Minecraft para poder jugar.

Otro gran inconveniente son las actualizaciones sobre este tipo de modificaciones, en ocasiones requiere de una actualización de la modificación de la parte cliente de Minecraft para poder disfrutar del juego.

Criterio de selección de la solución

Mi experiencia como administrador de un servidor modificado de Minecraft me dice que es preferible la solución del desarrollo de un complemento de Minecraft por las siguientes razones:

- La configuración del complemento está separada del servidor con una interfaz, que no todos los complementos suelen disponer, es un valor añadido de esta solución.
- Las actualizaciones de los complementos se pueden realizar técnicamente al vuelo y el jugador no se inmuta del cambio: apagas el servidor, incluyes el complemento actualizado, arrancas el servidor y el jugador se conecta normalmente. En cambio, el caso de la solución desarrollada mediante la modificación del servidor y del cliente, si la actualización abarca ambas partes (parte cliente y parte servidor), la accesibilidad del jugador al servidor queda subordinada al tiempo en el que el jugador descarga la actualización del cliente Minecraft y lo instala, ralentizando el proceso gravemente.
- Trabajar con código ofuscado es una tarea de comprensión e investigación agotadora, pero no hay más remedio si se quiere que el complemento no malgaste recursos, como ya se citó en la primera solución a la personalización de los monstruos de Minecraft en la que se capturaban los eventos de estas entidades para modificar su comportamiento en tiempo de ejecución.

5.3 Solución propuesta

La solución elegida, como se avanzó en el apartado anterior, será la del desarrollo de un complemento para un servidor no oficial de Minecraft.

Este proyecto seguirá la distribución estándar de un proyecto en cascada, con las siguientes fases:

- Gestión del proyecto: Esta fase se inició con el proyecto y terminará con él. En esta fase se hará un control de cambios sobre los obstáculos y oportunidades que se encuentren en cualquiera de las fases que a continuación se citarán.
- Fase 1 - Análisis e investigación: Esta fase trata la definición de objetivos, requisitos funcionales, no funcionales, de información, seguridad, herramientas e internacionalización.
- Fase 2 - Diseño de la solución: Esta fase establecerá las estrategias para abordar este proyecto, arquitectura del producto, diseño de las interfaces de la aplicación de escritorio y de los módulos del complemento.
- Fase 3 – Generación de código: Esta fase generará el producto software que se quiere alcanzar con este proyecto. Se dividirá en dos grandes bloques diferenciados: el complemento del servidor, que modelará los requisitos funcionales del complemento citados como casos de uso en el apartado del mismo nombre, y el desarrollo de una aplicación de escritorio para la creación/modificación de las configuraciones de los módulos en los que se separa el complemento.
- Fase 4 – Pruebas: Esta última fase servirá para validar el producto final obtenido y obtener resultados de este desarrollo.

Una característica principal de este proyecto es que gran parte de los módulos en los que se divide el complemento, están altamente acoplados con el de los jugadores, dado que el grueso de datos reside en él, los módulos de: grupos, clanes, clases, niveles, armas, armaduras, joyas, chat; dependen de las acciones de los jugadores. Por todo esto el desarrollo de este proyecto será de carácter modular.

Teniendo en cuenta el tamaño del proyecto es necesario un desarrollo en espiral, puesto que los nuevos módulos que se irán agregando al proyecto modificarán el tamaño de otros módulos, debido al alto acoplamiento entre ellos.

Además se hará uso de la estrategia *top-down*[22], dada la inexperiencia en el desarrollo de complementos de Minecraft y del desarrollo de videojuegos en general, de manera iterativa el grueso del proyecto aumentará con cada módulo agregado.

En detalle el desarrollo del complemento se dividirá en bloques que corresponden a los módulos anteriormente citados en el análisis de requisitos: módulo de jugadores, módulo de clases, módulo de niveles, módulo de chat, módulo de armas, módulo de armaduras, módulo de joyas, módulo de objetos, módulo de clanes, módulo de grupos, módulo de economía, módulo de monstruos y módulo de spawners. De igual manera, el desarrollo de la aplicación de escritorio se dividirá en módulos.

Una vez discernidos los datos necesarios para la creación de la configuración de cada módulo se comenzará con el desarrollo de la aplicación de escritorio.

5.4 Análisis de seguridad

En este apartado se relataran los aspectos de seguridad que conciernen al proyecto y que son de suma importancia.

Los servidores de Minecraft pueden categorizarse por un aspecto de su configuración que obliga o no a que sus jugadores tengan una cuenta oficial del juego adquirida,

[22] Enfoque top-down: enfatiza la planificación y el conocimiento completo del sistema.

reciben el nombre de servidores *Premium*[23] y servidores *no Premium*. Mientras que los servidores *Premium* permiten la conexión solo de jugadores con cuenta oficial adquirida, los servidores *no Premium* permiten la conexión no solo de los jugadores con cuenta adquirida, además permiten la conexión a los jugadores que no tienen una cuenta oficial adquirida. Lo cual puede llegar a ser un problema de seguridad para los jugadores, puesto que cualquier jugador puede cambiar su nombre de usuario, sea con cuenta oficial o sin ella, y suplantar a otro jugador.

Los complementos que manejan datos sobre jugadores, como puede ser un gestor de dinero, son vulnerables a estos tipos de ataques, por lo que es necesario un control sobre los identificadores de los jugadores. Las versiones más actuales de los servidores Minecraft incluyen identificadores únicos, denominados *UUID*[24], por jugador con cuenta oficial que diferencian un jugador de otro.

En esta primera versión de la solución solo se tendrán en cuenta los jugadores con cuenta oficial adquirida, de modo que este problema de seguridad está subsanado, pero limitado a servidores *Premium.*

Por otro lado, existen mods del cliente Minecraft que pueden ser usados de manera malintencionada, como por ejemplo mods que permiten ver a través de cualquier bloque, mods que detectan enemigos cerca y atacan automáticamente, y una gran lista interminable de ellos. A pesar de saber de su existencia y con intenciones futuras de poder evitar este tipo de ataques, no se tendrá en cuenta en este proyecto.

5.5 Análisis de eficiencia algorítmica

Los juegos MMORPG manejan grandes cantidades de datos en tiempo de ejecución (estadísticas de jugadores, datos de configuración, etcétera) por lo que es necesario una gestión eficiente de los datos del complemento que se ha desarrollado.

Cada acción que realiza el jugador, las máquinas de estado que definen el comportamiento de los *PNJs* (personajes del juego que no son jugadores) y otras entidades del juego disparan eventos que hacen uso de datos de diferentes orígenes y con los cuales se realizan cálculos estadísticos y deben ser obtenidos en tiempo real, es necesario entonces el uso de estructuras de datos de fácil y rápido acceso que permitan subsanar la demanda temporal que tiene esta índole de videojuegos.

En esta versión se ha tenido en cuenta la necesidad temporal de los datos, de manera que algunos de los datos (Sistema de niveles) son precargados en el arranque del servidor y manejados por estructuras de datos ya definidas por el lenguaje de programación Java que son eficientes de por sí. Se han desarrollado estructuras de datos específicas de cada tipo de objeto (Jugadores, Clanes, Clases, etcétera) que a su vez son encapsulados por otros tipos de estructuras de datos propios de Java para mejorar el tiempo de acceso y uso de los datos que encapsulan.

[23] Premium: calificación que reciben los servidores no oficiales que no permiten la conexión de jugadores que no posean una cuenta oficial de Minecraft.

[24] UUID: Identificador único y universal.

5.6 Análisis de la Internacionalización

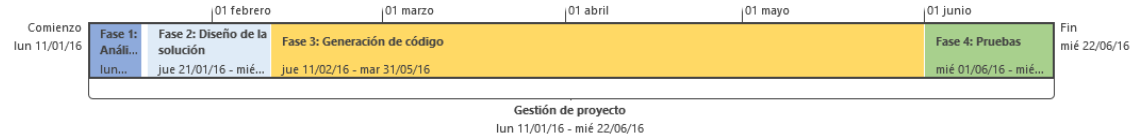
Con el fin de llegar a un mayor público los complementos desarrollados por la comunidad suelen poner a disposición de los administradores la posibilidad de cambiar el idioma de sus complementos, sobre todo en los casos que haya cierta interacción del jugador ante acciones con el complemento (cómo pueden ser mensajes de confirmación, advertencia, negación de permisos, etcétera).

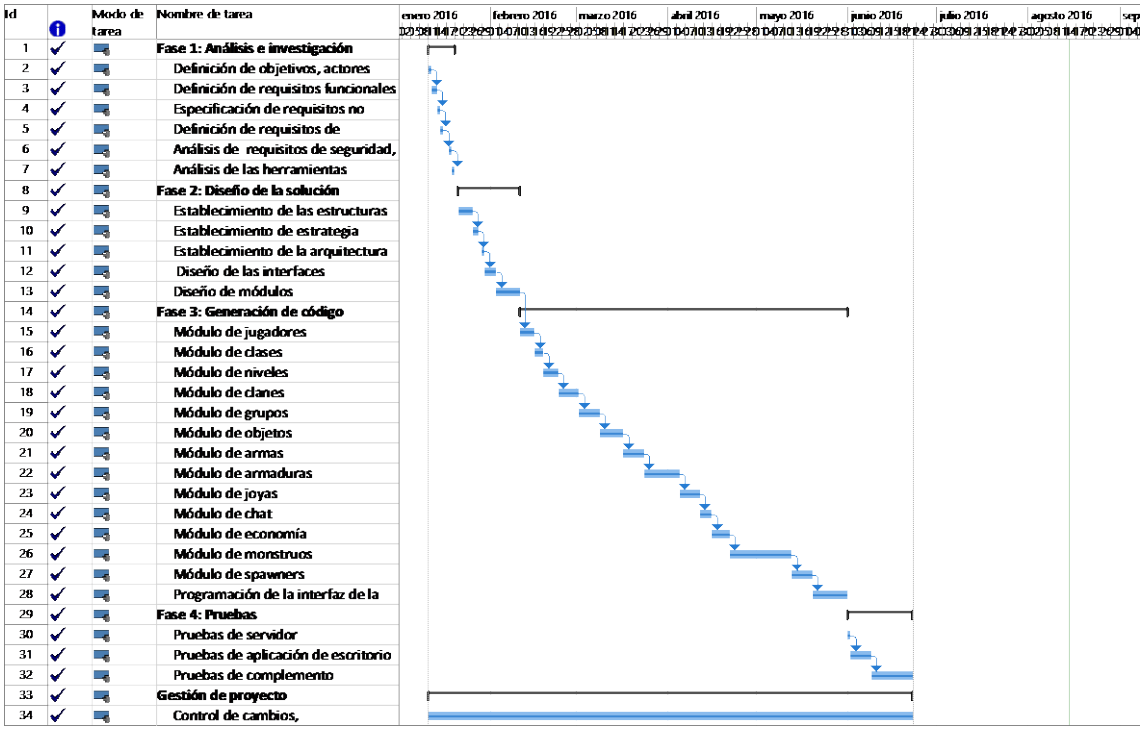
Dado que esta solución es la creación de un servidor de rol, mediante el desarrollo de un complemento, es de esperar que tenga los medios para realizar el cambio de idioma del mismo, pero dada la envergadura del proyecto ha sido rechazado este valor añadido y quedará como objetivo para trabajos futuros.

5.7 Presupuesto

En este apartado se hará una estimación de un presupuesto sobre los medios materiales y recursos humanos que se requerirán para el desarrollo de este proyecto.

Se ha realizado un Proyecto mediante MS Project [25]y se ha obtenido el siguiente diagrama de Gantt, especificando todas las fases con las que contará este proyecto y sus tareas dentro de cada fase.





[25] MS Project: Herramienta de administración de proyectos de Microsoft https://products.office.com/es-es/project/project-and-portfolio-management-software

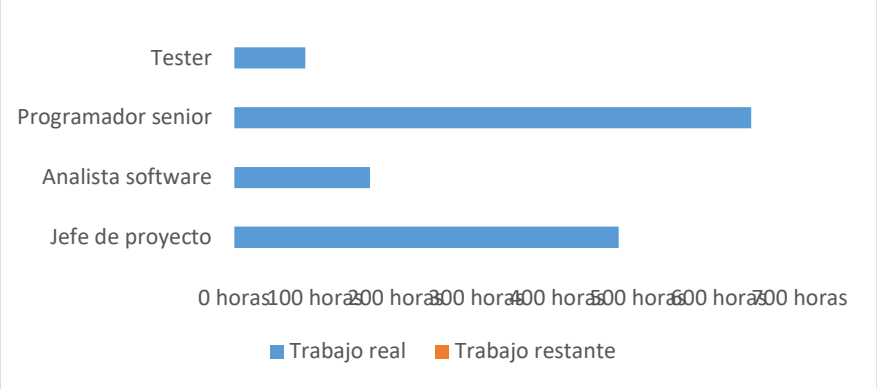
Se han estimado los costes materiales y de trabajo de los siguientes recursos:

- Jefe de proyecto técnico.
- Analista de software.
- Programador senior.
- *Tester*[26].
- Alquiler del hospedaje del servidor.
- Material hardware necesario para el desarrollo del proyecto.

Se incluyen para los recursos humanos precio por hora y el costo por día de uso, y de igual manera para los recursos materiales precio de compra y de costo por día de uso.

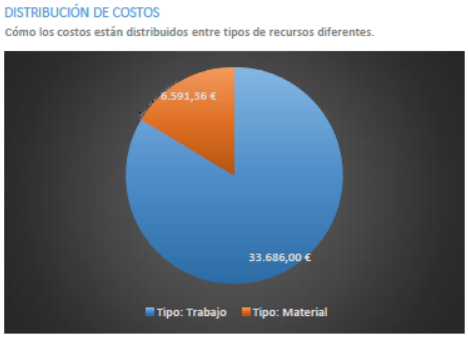
Id	Nombre del recurso	Tipo	Etiqueta de material	Iniciales	Grupo	Capacidad máxima	Tasa estándar	Tasa horas extra	Costo/Uso	Acumular	Calendario base	Código
1	Jefe de proyecto	Trabajo		JdP	Gestión de	50%	30,00 €/hora	0,00 €/hora	12,00 €	Prorrateo	Estándar	
2	Portátil programación	Material		P	Programaci		750,00 €		13,92 €	Comienzo		
3	Portátil de Jefe de proyecto	Material		P	Gestión de		750,00 €		13,92 €	Prorrateo		
4	Alquiler hospedaje servidor	Material	a.	A	Programaci		75,39 €		0,00 €	Prorrateo		
5	Portátil Tester	Material		P			750,00 €		13,92 €	Prorrateo		
6	Portátil Analista	Material		P	Diseño		750,00 €		13,92 €	Comienzo		
7	Analista software	Trabajo		A	Diseño	100%	30,00 €/hora	0,00 €/hora	10,00 €	Prorrateo	Estándar	
8	Programador senior	Trabajo		P	Programaci	100%	20,00 €/hora	0,00 €/hora	8,00 €	Prorrateo	Estándar	
9	Tester	Trabajo		T	Pruebas	100%	15,00 €/hora	0,00 €/hora	5,00 €	Prorrateo	Estándar	

Se han obtenido un total de 1372 horas para el desarrollo completo de este proyecto, que son divididos entre los distintos recursos humanos de los que se han hecho uso.



El costo del proyecto se puede ver en detalle en los siguientes gráficos. Cómo se puede apreciar el costo principal reside en los recursos humanos usados para este proyecto, mientras que los gastos materiales quedan en un segundo lugar.

[26] Tester: en español Probador es el encargado de notificar los errores y bugs encontrados en el videojuego en las fases de pruebas de integración.

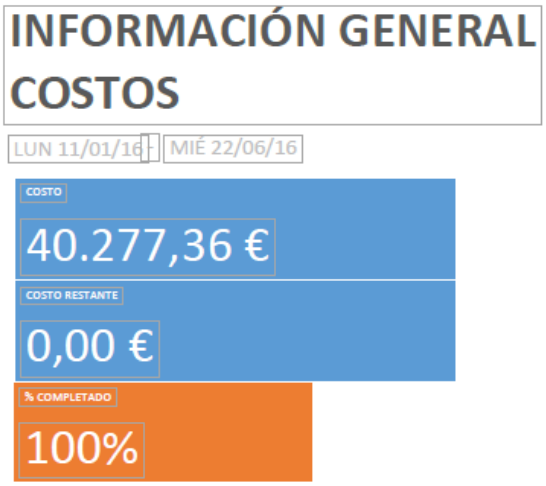


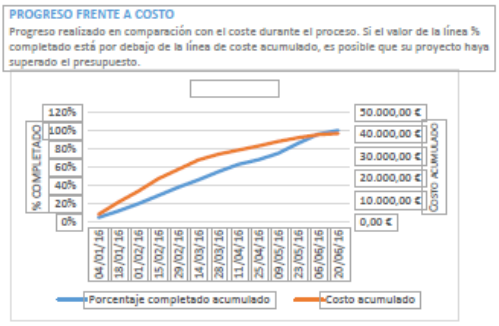
DETALLES DE COSTOS

Detalles de costos de todos los recursos de trabajo.

Nombre	Trabajo real	Costo real	Tasa estándar
Jefe de proyecto	476 horas	14.286,00 €	30,00 €/hora
Analista software	168 horas	5.150,00 €	30,00 €/hora
Programador senior	640 horas	12.920,00 €	20,00 €/hora
Tester	88 horas	1.330,00 €	15,00 €/hora

El presupuesto total del proyecto es de 40.277,36 €, en los siguientes gráficos se puede visualizar los costos en cada una de las fases del desarrollo del proyecto.





ESTADO DEL COSTO

Estado de costo de tareas de nivel superior.

Nombre	Costo real	Costo restante	Costo de línea base	Costo	Variación de costo
Fase 1: Análisis e investigación	2.503,92 €	0,00 €	0,00 €	2.503,92 €	2.503,92 €
Fase 2: Diseño de la solución	4.849,76 €	0,00 €	0,00 €	4.849,76 €	4.849,76 €
Fase 3: Generación de código	15.779,84 €	0,00 €	0,00 €	15.779,84 €	15.779,84 €
Fase 4: Pruebas	2.093,92 €	0,00 €	0,00 €	2.093,92 €	2.093,92 €
Gestión de proyecto	15.049,92 €	0,00 €	0,00 €	15.049,92 €	15.049,92 €

6 Diseño de la solución

En este apartado se realizará la fase de diseño de la solución propuesta para este TFG. Para empezar se hará un análisis sobre las posibles herramientas en las que se apoyará este proyecto y se hará una comparativa sobre cuáles son las más competentes para su realización, se seguirá con la arquitectura del software en el que se mostrará un diagrama de clases general con la distribución de los módulos y unas pocas

dependencias del complemento y la aplicación de escritorio. Para acabar, se diferenciarán ambas partes anteriormente citadas y se subdividirán en los módulos que los componen con descripciones detalladas de sus funciones, diagramas de clases y diagramas de secuencia si son necesarios.

6.1 Análisis de las herramientas

En este apartado se va a realizar un análisis de las posibles herramientas que se utilizarán mostrando los pros y contras de cada tecnología usada.

6.1.1 Descripción general

Minecraft es un videojuego que ha tenido su auge tanto para computador como para videoconsolas, en este caso este trabajo solo hace hincapié en la versión para computador. Desde el punto de vista del computador, la versión servidor de Minecraft es multiplataforma (Linux, OS X y Windows) y está desarrollado en el lenguaje de programación en Java. Además cabe destacar que, los servidores no oficiales (Spigot, CraftBukkit, Sponge, etcétera), cuentan con su propia API de programación de complementos con la cual se pueden desarrollar los complementos que pueden añadir nuevas funcionalidades al servidor de Minecraft.

6.1.2 Entorno de desarrollo de aplicaciones

Para el desarrollo de la solución se ha barajado las opciones para elegir un entorno de desarrollo eficaz, de entre los conocidos: Netbeans[27], Eclipse[28] y IntelliJ[29]. A continuación una tabla resumen de las características que ofrecen y son necesarias para el proyecto.

	Interfaz	**Plugins propios del IDE**	**Herramientas internas**	**Experiencia previa**	**Precio**
Netbeans	Amigable, Organizada	Oficiales y No Oficiales, pueden no recibir actualizaciones	Soporte a equipos, Herramientas de asistencia a la refactorización, Autocompletado de código	Alta	Gratuito
Eclipse	Simple, Organización de menús contextuales caótica	Oficiales y No Oficiales, pueden no recibir actualizaciones	Refactorización escasa, Autocompletado de código semiautomático, Soporte a equipos	Media	Gratuito

[27] Netbeans: https://netbeans.org/

[28] Eclipse: https://eclipse.org/

[29] IntelliJ: https://www.jetbrains.com/idea/

IntelliJ	Amigable (similar a Netbeans), Organizada	Oficiales propios de los creadores siempre actualizados.	Refactorización y autocompletado de código según la configuración, Soporte a equipos	Baja	Periodo de prueba de 30 días, en definitiva de pago

Se hará uso del entorno de desarrollo integrado Netbeans, el cual es una solución completa para el desarrollo de aplicaciones en el lenguaje de programación Java, la creación de interfaces resulta amigable, ofrece soporte a repositorios y tengo la suficiente experiencia adquirida a lo largo del grado como para hacer un uso eficiente de sus herramientas internas de refactorización.

6.1.3 Sistema de ficheros

El aspecto multiplataforma del videojuego, es heredado de Java por lo que no supone un mayor problema, siempre que se tenga en cuenta el tipo de sistema de ficheros con el que trabaja cada sistema operativo. Este punto se tendrá en especial cuenta a la hora de gestionar los ficheros de configuración del complemento generados por la interfaz.

6.1.4 Interfaz de programación de aplicaciones

En cuanto a la interfaz de programación de aplicaciones de Minecraft, dada la poca experiencia con la que he contado desarrollando complementos básicamente de gestión en mi propio servidor, he realizado un estudio intensivo sobre las funcionalidades y posibilidades que ofrece para el desarrollo de la solución. Mediante la propia API que ofrecen los desarrolladores de las versiones oficiales. A continuación una breve comparativa entre las posibles interfaces de programación de aplicaciones para servidores no oficiales de Minecraft.

	Disponibilidad del servidor	**Disponibilidad del Javadoc del API**	**Actualizaciones**	**Compatibilidad de plugins entre APIs**
Spigot API	Disponible	Disponible	Recibe actualizaciones	Depende de las versiones en que se han desarrollado los plugins
Bukkit API	No disponible los enlaces de descarga han sido retirados	Disponible	No recibe actualizaciones desde el aviso de cese por infringir la ley de derechos de autor de la era digital (DMCA)	Poca, desde el aviso de cese muchos plugins han dejado de ser actualizados y han pasado a desarrollarse

				con la API de Spigot

Se hará uso de la API Spigot por las dificultades que pudiera encontrar para desarrollar un complemento con la API de Bukkit. El no poder descargar desde el sitio oficial Bukkit la versión del servidor (no oficial) de Minecraft es cuestión indispensable para elegir la opción Spigot.

6.1.5 Servidor no oficial de Minecraft

Como se avanzó en líneas anteriores la disponibilidad de tener el servidor a mano es necesaria, puesto que como se vio en el análisis de requisitos del complemento va a ser necesario poder modificar los atributos de las entidades con tal de poder mejorarlas. En este caso, y dado que el resto de *mods* bien no están suficientemente desarrollados o bien no están disponibles, se usará la versión 1.8.7 del servidor Minecraft ofrecida por Spigot. Esta versión, la cual uso en mi propio servidor, incluye casi todas las mejoras de la actual última versión (Minecraft 1.10) y tiene, al menos junto con la API, lo necesario para el desarrollo del complemento.

Para las pruebas unitarias se hará uso de un servidor local sobre un sistema operativo Windows 10, mientras que para las pruebas de todo el sistema en conjunto se hará uso de un hospedaje alquilado para el servidor sobre sobre un sistema operativo Linux.

6.2 Arquitectura software

En este apartado se hará un esbozo de la arquitectura del complemento para servidor Minecraft con diagramas de clases y de secuencia en el caso que haga falta.

El diseño de la solución se dividirá en dos bloques que serán la aplicación de escritorio y el complemento, y en ellos los distintos módulos que los componen. Debido al tamaño del proyecto es imposible mostrar los módulos con los atributos y métodos de cada clase sin que ocupen varias páginas, lo que dificultaría su lectura.

Hay que remarcar que cada complemento desarrollado para servidores no oficiales de Minecraft sigue el patrón de diseño *Singleton* [30]no permitiendo así más que una instancia de un mismo complemento en el servidor.

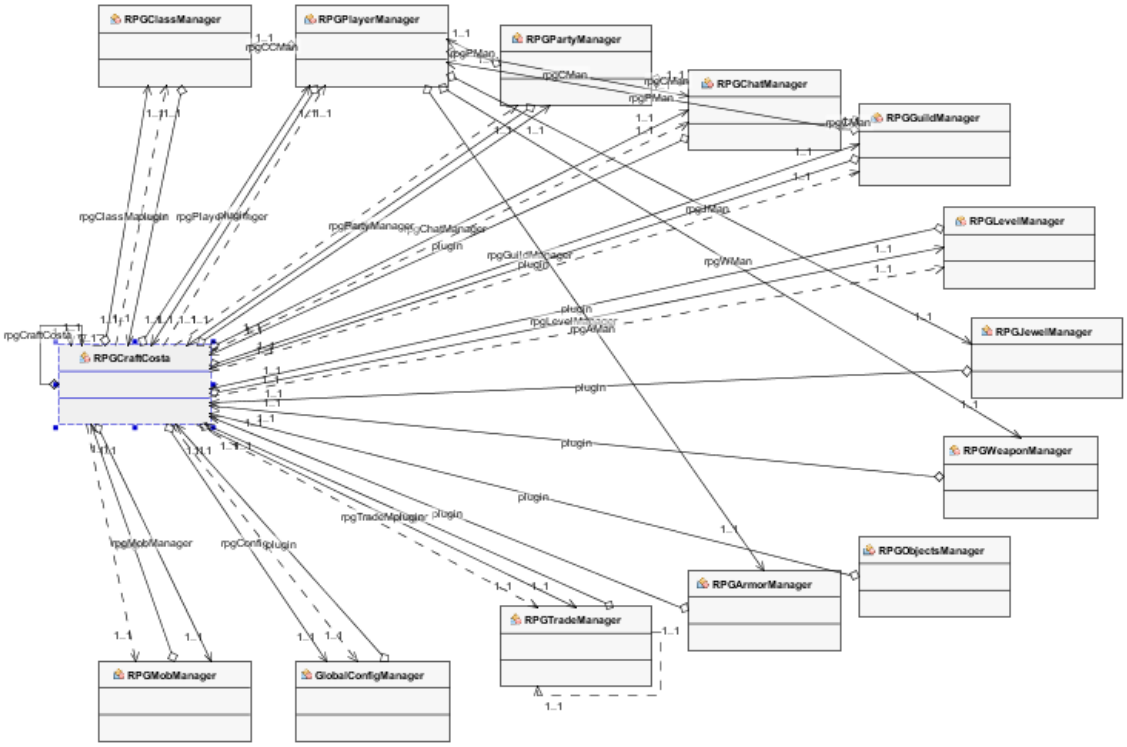
6.2.1 Diseño del complemento

Los módulos del complemento irán acompañados por una descripción, la información que manejan, diagramas de clase explicativos y diagramas de secuencia en el caso de que sea necesario. Cada módulo tendrá un gestor, un oyente que escuche por los eventos a los que esté relacionados, una clase de comandos (la básica para que los usuarios puedan interactuar con el complemento) y las clases que sean necesarias para que el módulo esté completo.

[30] Singleton: Patrón de diseño que solo permite la ejecución de una sola instancia al mismo tiempo.

En todo complemento de Minecraft existe una clase que extiende del paquete *JavaPlugin*[31] (Cómo se puede ver en el anexo II), esta clase no es más que el cargador del complemento en el servidor que ordena directrices al servidor. Cuando se avanzó que el proyecto está altamente acoplado hacía referencia a que habrá una instancia en cada gestor, una clase que capture los eventos y una clase de comandos, con tal de poder comunicar todos los módulos mediante una petición a la clase principal.

Debido al tamaño total de la parte del complemento, es imposible mostrar todo el diagrama de clases por lo que se ha reducido a complemento y las relaciones con los gestores de cada módulo:



6.2.1.1 Módulo de jugadores

Este módulo se encarga de los manejar las acciones y datos de los jugadores. Los objetivos de este módulo son:

- Crear datos de los jugadores si no existen o cargar los datos en caso contrario.
- Actualizar la información del jugador.
- Guardar la información de los jugadores.

Se compone de las siguientes clases

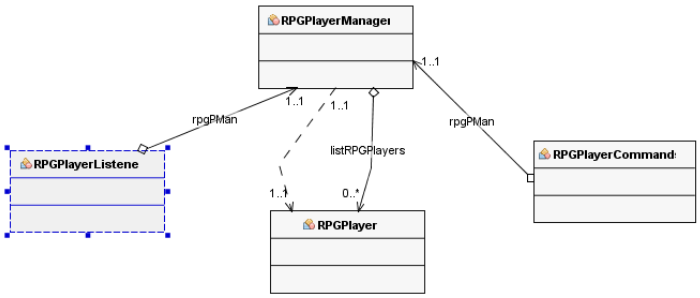
- ***RPGPlayerManager***: Gestor de jugadores realiza las actualizaciones de los atributos de los jugadores.
- ***RPGPlayerListener***: Captura eventos del jugador para cargar, crear o guardar sus datos. Los posibles eventos que disparan las acciones anteriores son: el

[31] JavaPlugin es la clase principal que deben extender los complementos para que servidor de Minecraft cargue los complementos.

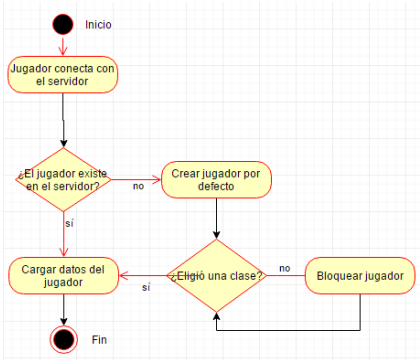
jugador se conecta, pierde la conexión, es echado por un administrador, desconecta del servidor.

- ***RPGPlayerCommands***: Captura los comandos del jugador al hacer consultas sobre sus datos.
- ***RPGPlayer***: Clase modelo para la creación de jugadores, contiene los atributos y las herramientas para consultarlos y modificarlos. En detalle contiene los siguientes datos: nombre, *UUID*, clase, clan, grupo, dinero, experiencia, nivel, atributos propios de su clase, variables de gestión del chat.

El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



Cuando un jugador se conecta por primera vez al servidor, se crean por defecto los atributos del jugador y se evita que se mueva hasta que haya elegido una clase, el siguiente diagrama de actividades lo resume:



Cuando un jugador se desconecta sea la razón que sea se guardan sus datos. Para cada jugador los datos son guardados en ficheros individuales de formato Yaml con su *UUID* cómo título en una carpeta en la configuración del complemento.

6.2.1.2 Módulo de clases

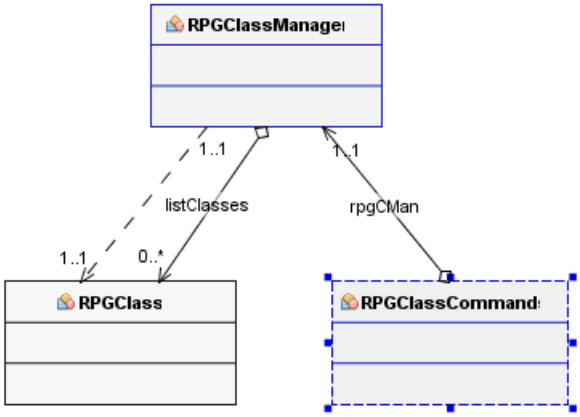
Este módulo se encarga de gestionar las distintas clases que existan en la configuración del servidor. Los objetivos de este módulo son:

- Cargar las distintas clases.
- Permitir la elección de clases a los jugadores.
- Permitir listar y visualizar los datos de las clases.
- Permitir subir los atributos a los jugadores,

Se compone de las siguientes clases:

- ***RPGClassManager***: Gestiona la carga de clases del complemento.
- ***RPGClassCommands***: Captura los comandos del jugador para permitir la elección, listar clases y visualizar sus atributos.
- ***RPGClass***: Clase modelo para la creación de clases, contiene atributos y herramientas para obtenerlos y visualizarlos.

El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



Los datos sobre las clases son almacenados en un único fichero de formato Yaml en la carpeta de configuración del complemento.

6.2.1.3 Módulo de niveles

Este módulo se encarga de gestionar los niveles. Los objetivos de este módulo son:

- Cargar los datos sobre la configuración de niveles y crear el árbol de niveles.
- Comprobar si los jugadores suben de nivel, para comunicarlo al gestor de clases.
- Ofrecer herramientas para obtener estadísticas sobre experiencia

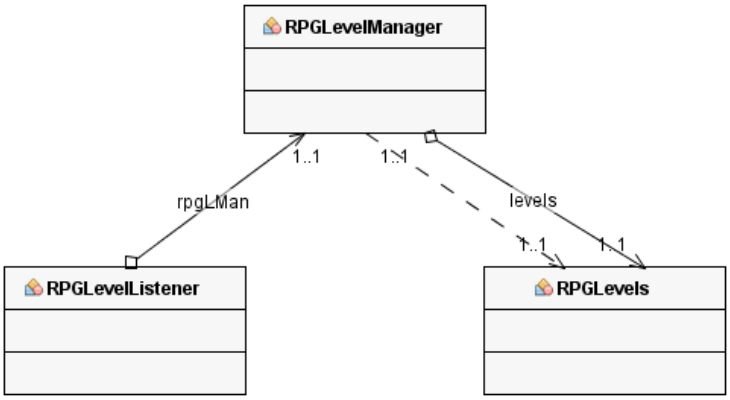
Las clases involucradas en este módulo son:

- ***RPGLevelManager***: Clase que gestiona la carga de la configuración sobre niveles, crea el árbol de niveles y ofrece herramientas de consulta sobre el nivel de un jugador dependiendo de la experiencia.
- ***RPGLevel***: Clase modelo que almacena la configuración de los niveles. Entre sus datos constan: variables de la fórmula de cálculo, el máximo nivel alcanzable y el árbol de niveles.
- ***RPGLevelListener***: Clase que captura los eventos que influencian directamente en la experiencia de un jugador. En este caso se captura la acción cuando un jugador mata a un monstruo, añade la experiencia al jugador y calcula si habrá un cambio de nivel, en el caso de que esto último ocurra el módulo de clases deberá actualizar los atributos del jugador.

El árbol de niveles debe ser eficaz en cuanto a búsquedas de nivel según la experiencia del jugador por lo que es necesario una estructura de datos tipo mapa clave-valor que

permita la búsqueda rápida de niveles. En este caso se hará uso de un *TreeMap*[32] con clave experiencia y valor nivel que permita obtener el nivel a partir de una experiencia. Una ventaja de ésta estructura de datos es que también permite obtener el próximo nivel y la experiencia requerida para ello, por lo que se usará para calcular la experiencia necesaria para el siguiente nivel y el porcentaje de experiencia acumulado.

El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



6.2.1.4 Módulo de chat

Este módulo se encarga de gestionar los chats o canales del servidor. Los objetivos de este módulo son:

- Cargar la configuración de los chats o canales desde el fichero de configuración.
- Capturar los eventos de chats y darles formato según el canal o chat al que va dirigido.
- Permitir al jugador elegir que chats o canales quiere leer.
- Delegar el envío de mensajes de clan, grupo o mercado a sus propios gestores.

Las clases que forman este módulo son:

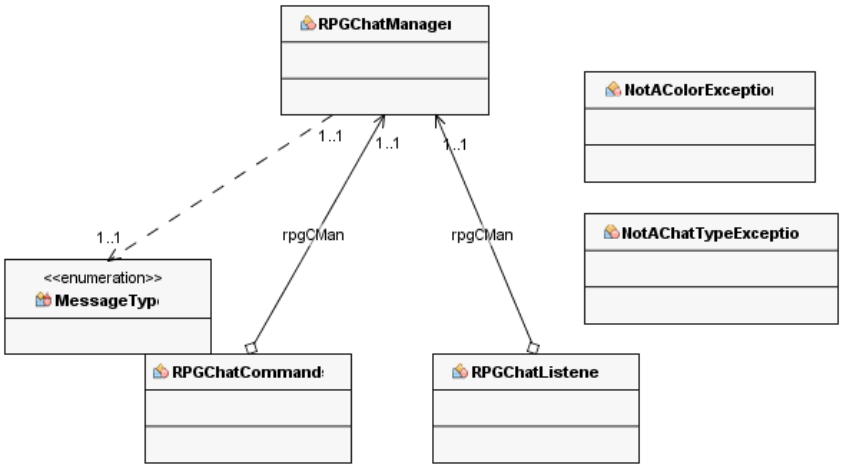
- ***RPGChatManager***: Clase que gestiona la carga de la configuración de los chats o canales. Contiene los datos de configuración global de los chats y ofrece herramientas para la consulta de los datos de configuración y métodos de cálculo para distancias entre jugadores de un mismo canal.
- ***RPGChatListener***: Clase que captura los eventos de chat de los jugadores o administradores que formatea los mensajes y los delega en otros módulos si pertenecen a clanes, grupos o mercado.
- ***RPGChatCommand***: Clase que ofrece herramientas de gestión de chats o canales a los jugadores, pudiendo habilitar o deshabilitar los chats.
- ***MessageType***: Enumerado en el que se encuentran los distintos tipos de chats y sus formatos por defecto. Al cargar la configuración si esta es distinta a la del enumerado es modificada. Los distintos tipos de chats son:

[32] TreeMap: es una estructura de datos ordenada de formato clave-valor que permite búsquedas rápidas.

- Local: Canal de mensajes limitado por la proximidad de los oyentes y de su localización en un mismo mundo.
- Global: Canal de mensajes que se envían a todo el servidor independientemente del mundo en el que se encuentre.
- Mercado: Canal de mensajes que se envían independientemente del mundo en el que se encuentre a todo el servidor para realizar una transacción comercial entre jugadores.
- Clan: Canal de mensajes restringidos a los jugadores del clan al que pertenece el jugador que envía un mensaje.
- Grupo: Canal de mensajes restringidos a los componentes del grupo al que pertenezca el jugador que envió el mensaje.
- Privado: Canal de mensajes privados entre dos jugadores.
- Noticias: Canal de mensajes de difusión global que muestra noticias sobre el servidor usado por los administradores.
- Alarmas: Canal de mensajes críticos del servidor, usado por los administradores para indicar incidencias.

- ***NotAColorException*** y ***NotAChatTypeException***: Son clases que extienden las Excepciones de Java para mostrar errores en la configuración de chats.

El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



6.2.1.5 Módulo de clanes

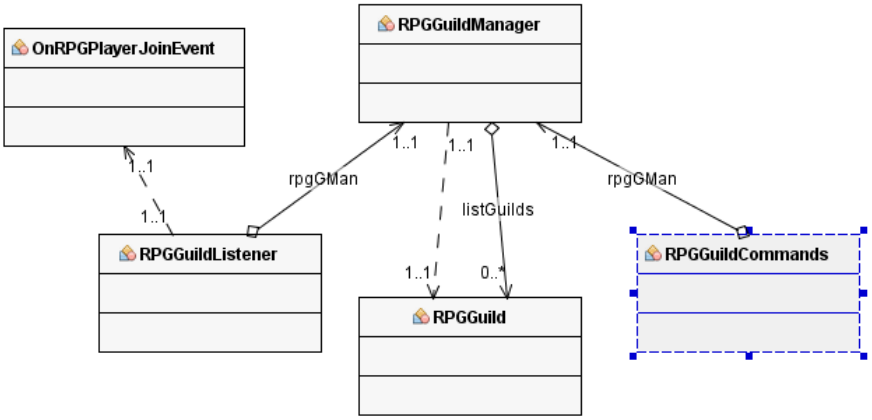
Este módulo se encarga de cargar la configuración de los clanes y los clanes existentes en el servidor. Sus objetivos son:

- Cargar la configuración de clanes.
- Cargar los clanes existentes almacenados.
- Controlar los miembros de los clanes.
- Controlar las peticiones/invitaciones de unión a los clanes.
- Ofrecer herramientas a los jugadores para administrar un clan.
- Envío de mensajes dentro de un mismo clan.

Las clases que intervienen en este módulo son:

- ***RPGGuildManager***: Carga la configuración de clanes y los clanes existentes en el servidor.
- ***RPGGuildCommands***: Clase que ofrece herramientas de gestión para los propietarios y miembros de los clanes.
- ***RPGGuildListener***: Clase que captura los eventos de jugadores que tienen relación con un clan. Estos eventos son conexión/desconexión (***OnRPGPlayerJoinEvent***, ***PlayerQuitEvent***, ***PlayerKickEvent***) al servidor, si el clan está basado en niveles y opción de contribución por muertes de monstruos también estará el evento de muerte de entidades (***EntityDeathEvent***).
- ***RPGGuild***: Clase modelo que almacena la información de los clanes, entre sus datos se encuentran: Nombre del clan, propietario, listas de miembros, listas de miembros en línea, nivel del clan y dinero.
- ***OnRPGPlayerJoinEvent***: Esta clase representa un evento que se dispara en el momento en el que un jugador de un clan se conecta al servidor.

El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



6.2.1.6 Módulo de grupos

Este módulo se encarga de cargar la configuración de los grupos, a diferencia de los clanes, los grupos no son almacenados en ficheros dado que son de carácter temporal. Sus objetivos son:

- Cargar la configuración de grupos.
- Ofrecer herramientas de gestión a los jugadores pertenecientes a un grupo.
- Controlar las peticiones/invitaciones de unión a los grupos.
- Controlar a los componentes de los equipos.
- Envío de mensajes dentro de un mismo grupo.

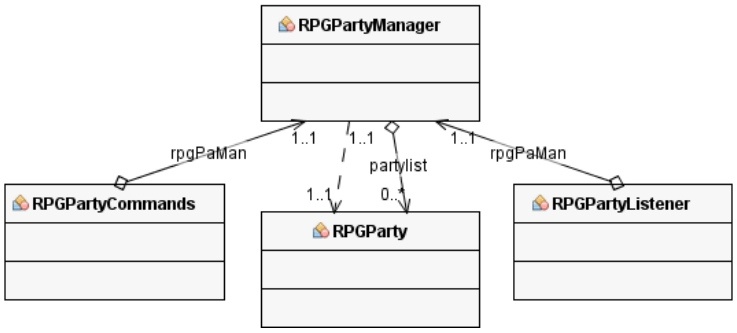
Las clases que intervienen en este módulo son:

- **RPGPartyManager**: Clase gestor del módulo de grupos se encarga de cargar la configuración de los grupos y mantener los grupos en el servidor.
- ***RPGPartyListener***: Clase que captura los eventos de jugadores que tienen relación con los grupos. Entre los eventos se encuentran: Desconexión del

jugador (***PlayerQuitEvent***, ***PlayerKickEvent***) y si la configuración de grupos contiene algún tipo de reparto hará efecto a la muerte de los monstruos (***EntityDeathEvent***).

- ***RPGPartyCommands***: Clase que ofrece herramientas de gestión a los líderes y miembros de los grupos.
- ***RPGParty***: Clase modelo que almacena la información de los grupos. Entre sus datos se encuentran: Nombre del grupo, líder del grupo, miembros y muertes por miembro (este último solo es necesario si está configurado un sistema de reparto).

El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



6.2.1.7 Módulo de armaduras

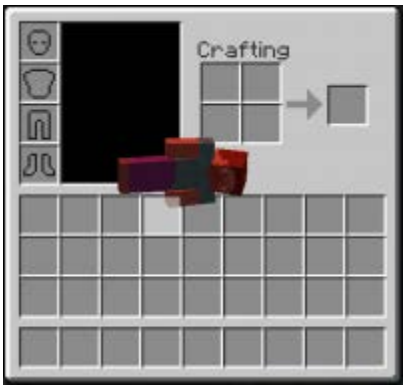
Este módulo se encarga de cargar la configuración de armaduras y las armaduras existentes en el servidor. Sus objetivos son:

- Cargar la configuración de las armaduras
- Cargar las armaduras en el servidor.
- Controlar el equipo de los jugadores.
- Permitir mejorar partes de la armadura.

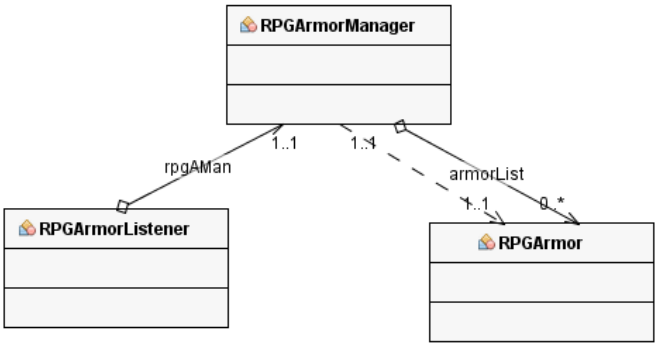
Las clases que forman este módulo son:

- ***RPGArmorManager***: Clase encargada de la carga de la configuración y de armaduras en el servidor. La configuración de armaduras consiste en crear un objeto que permitirá la mejora de una parte de una armadura mediante la combinación del objeto con la parte de la armadura. En detalle el objeto mejorador podrá romper, bajar el nivel, no tener efecto o subir el nivel de la armadura según con las probabilidades con las que se haya configurado.
- ***RPGArmor***: Clase modelo que representa una parte de la armadura. Entre los datos que maneja se encuentran: nombre, parte de la armadura, nivel, nivel de la armadura, atributos propios de una armadura. Está restringida a las armaduras que actualmente existen en Minecraft.
- ***RPGArmorListener***: Clase que captura los cambios realizados por el jugador al equipar/quitar partes de la armadura de su inventario. Para equipar una armadura es necesario capturar los eventos sobre el inventario del jugador de manera que si se equipa o quita una parte de la armadura añada/quite los atributos que la parte confiere a su portador. A continuación una captura de cómo es el inventario del jugador. Como se puede apreciar en el lado superior

izquierdo se aprecian 4 cuadrados en los que se puede equipar la armadura. Esta clase capturará las acciones sobre esos 4 cuadrados con tal de manejar las armaduras de los jugadores.



El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



6.2.1.8 Módulo de armas

Este módulo se encarga de cargar la configuración de armas y las armas existentes en el servidor. Sus objetivos son:

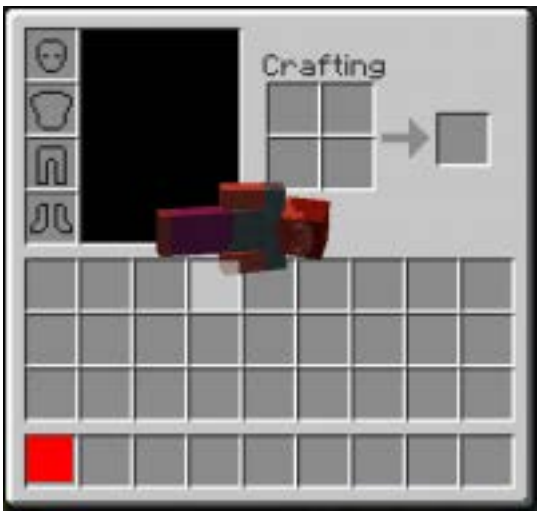
- Cargar la configuración de las armas
- Cargar las armas en el servidor.
- Controlar las armas que porten los jugadores.
- Permitir mejorar armas.

Las clases involucradas en este módulo son:

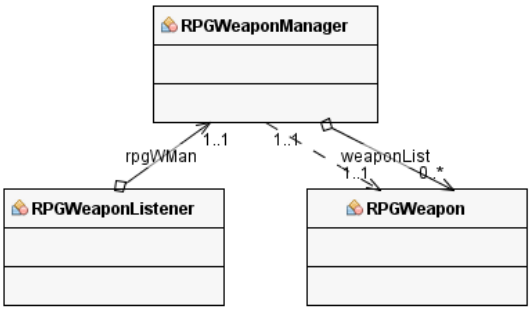
- ***RPGWeaponManager***: Clase encargada de la carga de la configuración y de armas en el servidor. La configuración de armas consiste en crear un objeto que permitirá la mejora de un arma mediante la combinación del objeto con el arma. En detalle el objeto mejorador podrá romper, bajar el nivel, no tener efecto o subir el nivel del arma según con las probabilidades con las que se haya configurado.
- ***RPGWeapon***: Clase modelo que representa un arma. Entre los datos que maneja se encuentran: nombre, objeto que representa un arma, nivel, nivel del arma, atributos propios del arma. Está restringida a las armas que actualmente

existen en Minecraft, aunque es una posibilidad crear armas con otros objetos no destinados a ello.

- ***RPGWeaponListener***: Clase que captura los cambios realizados por el jugador al equipar/quitar un arma del equipamiento. Para equipar un arma es necesario capturar los eventos sobre el inventario del jugador de manera que si se equipa o quita un arma añada/quite los atributos que la parte confiere a su portador. A continuación una captura de cómo es el inventario del jugador. A diferencia de las armaduras, las armas no tienen un cuadrado propio para su equipamiento, por lo que se ha preferido crear uno fijo para que realice esta función. Como se puede apreciar en el lado inferior izquierdo se aprecia 1 cuadrado marcado de color rojo, el cual se hará uso para equipar armas. Esta clase capturará las acciones sobre ese cuadrado con tal de manejar las armas de los jugadores. Cabe destacar que esa última barra en el inventario se llama *HotBar*[33] que se emplea en el juego como barra de acceso rápido al inventario y es posible que se pueda usar toda la barra para que realice la función de equipamiento del jugador.



El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



[33] HotBar: o barra de acceso rápido en el entorno de videojuegos facilita al acceso a una pequeña parte del inventario del jugador.

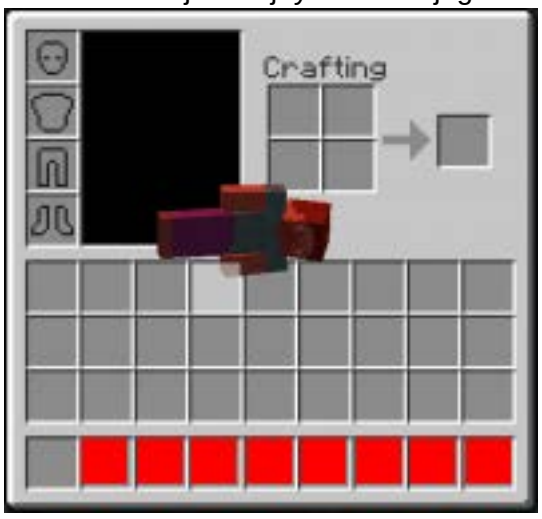
6.2.1.9 Módulo de joyas

Este módulo se encarga de cargar la configuración de joyas y las joyas existentes en el servidor. Sus objetivos son:

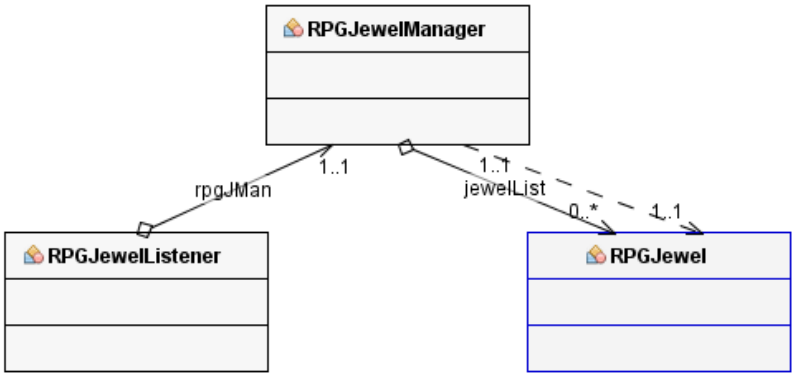
- Cargar la configuración de las joyas
- Cargar las joyas en el servidor.
- Controlar las joyas que porten los jugadores.
- Permitir combinar joyas.

Las clases involucradas en este módulo son:

- ***RPGJewelManager***: Clase encargada de la carga de la configuración y de joyas en el servidor. La configuración de joyas consiste en 4 probabilidades que marcarán los resultados de combinar 2 joyas del mismo nombre. En detalle estas 4 probabilidades corresponden a:
  - Probabilidad de rotura de ambas joyas
  - Probabilidad de pérdida de un solo atributo de la joya resultante.
  - Probabilidad de pérdida de más de un atributo de la joya resultante.
  - No tener efecto, en este caso la joya resultante es una de las dos y la otra rompe.
  - Probabilidad de éxito en la que se combinan los atributos de ambas joyas en una sola.
- ***RPGJewel***: Clase modelo que representa una joya. Entre los datos que maneja se encuentran: nombre, objeto que representa la joya, atributos propios de la joya. No está restringida a ningún objeto en especial.
- ***RPGJewelListener***: Clase que captura los cambios realizados por el jugador al equipar/quitar un joya del equipo. Para equipar una joya es necesario capturar los eventos sobre el inventario del jugador de manera que si se equipa o quita una joya añada/quite los atributos que confiere a su portador. A continuación una captura de cómo es el inventario del jugador. A diferencia de las armaduras e igual que las armas, no tienen un cuadrado propio para su equipamiento, por lo que se ha preferido usar la parte restante de la *HotBar* para esa función. Como se puede apreciar en el lado inferior los cuadrados rojos representan los huecos en los cuales se podrá equipar una joya. Esta clase capturará las acciones sobre estos cuadrados con tal de manejar las joyas de los jugadores.



El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



6.2.1.10 Módulo de economía

Este módulo se encarga de ofrecer las herramientas a los jugadores para que realicen transacciones comerciales tanto de dinero como de objetos, ya sean armaduras, armas, joyas u objetos de otra índole. En el juego original no existe el dinero y los intercambios de objetos con otros jugadores se realizan mediante cofres o soltando los objetos al suelo para que el otro jugador los recoja, por lo que es necesario crear las herramientas para realizar intercambios seguros entre 2 jugadores, dado que si se usan los medios originales cualquier otro jugador podría aprovecharse de esa ventaja y robar los objetos.

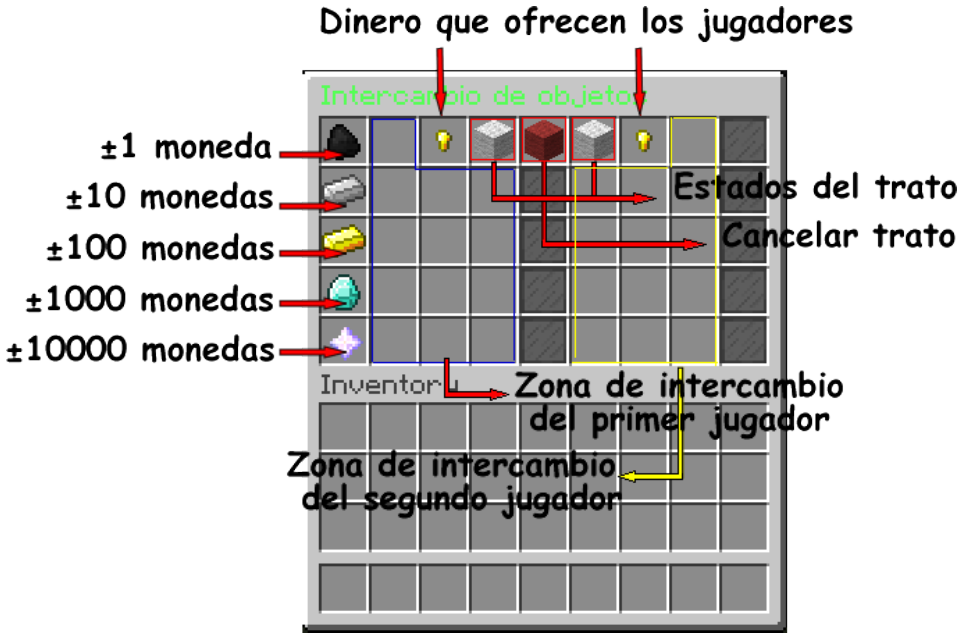
Los objetivos de este módulo son:

- Crear un sistema monetario para que los jugadores puedan almacenar dinero.
- Crear un sistema de intercambio de objetos y monetario seguro para las transacciones.
- Ofrecer herramientas a los jugadores para que realicen transacciones comerciales.

Las clases que intervienen en este módulo son:

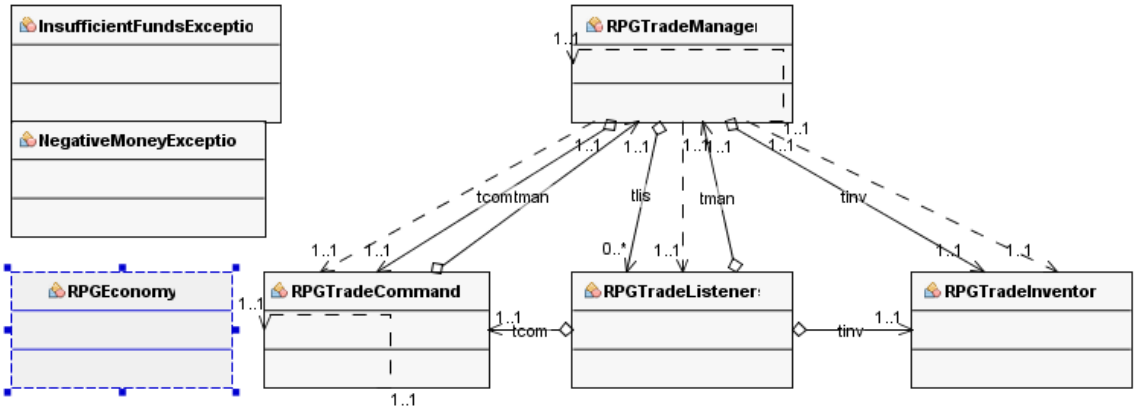
- ***RPGEconomy***: Clase modelo que representa el dinero de un jugador. Cada jugador tiene una instancia de esta clase entre sus atributos. Almacena el dinero del jugador y ofrece herramientas para su modificación.
- ***RPGTradeManager***: Clase que gestiona las transacciones mediante las clases que se verán a continuación.
- ***RPGTradeCommands***: Clase que ofrece herramientas a los jugadores para iniciar transacciones entre ellos.
- ***RPGTradeInventory***: Clase modelo de inventario que diseña la interfaz del inventario de intercambio. En la siguiente imagen se puede apreciar la estructura del inventario de intercambio. Este inventario modificado se mostrará a los dos jugadores cuando inicien un trato. La parte inferior mostrará el inventario propio del jugador y la parte superior el inventario de intercambio. En la parte superior, el lateral izquierdo corresponderá al iniciador del intercambio y el derecho al otro jugador, cualquier cambio de los objetos será visible a ambos jugadores. La columna de objetos que aparecen a la izquierda sirven para añadir dinero al

intercambio de manera fácil y rápida mediante el botón izquierdo y derecho del ratón. Las pepita de oro al añadir/retirar dinero muestra en su leyenda el dinero que ofrecen los participantes. En la parte central se muestran 3 bloques, el del medio cancela el trato y los de los laterales sirven para que el jugador vea el estado del trato o si es de su lado acepte el trato. Los estados de un trato pueden ser: No aceptado, Aceptado/No confirmado y Confirmado. Para evitar estafas el sistema de aceptación del trato no permite cambios de objetos o dinero, por lo que cualquier acción antes de aceptar un trato modificará los estados a no aceptado. En la parte central la columna oscura es la frontera que divide los inventarios de intercambio de ambos jugadores, por lo que cada uno podrá modificar su lado.



- ***RPGTradeListeners***: Clase que captura los eventos del inventario y de desconexión del jugador. Este captador de eventos crea la dinámica de intercambio de objetos mediante la actualización del inventario que ve cada jugador dentro de una transacción. En caso de pérdida de conexión los objetos y dinero volverán a sus propietarios.
- ***InsufficientFoundsException*** y ***NegativeMoneyException***: Clases que representan excepciones necesarias para el control de intercambios comerciales.

El diagrama de clases del módulo es el siguiente, por limpieza los atributos y métodos de las clases han sido ocultados.



6.2.1.11 Módulo de objetos

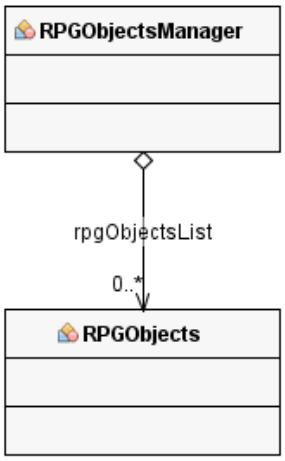
Este módulo se encarga de carga de los objetos personalizados para el servidor. Sus objetivos son:

- Cargar los objetos al servidor.

Las clases que intervienen en este módulo son:

- ***RPGObjectsManager***: Clase encargada de la carga de objetos.
- ***RPGObjects***: Clase modelo que representa un objeto. Entre sus datos se encuentran: nombre del objeto y una lista de descripciones. El uso que tienen será el propio que tienen en el juego original, como puede ser la comida para alimentarse.

El diagrama de clases de este módulo es el siguiente, por limpieza los atributos y métodos han sido ocultados.



6.2.1.12 Módulo de monstruos y spawners

Este módulo se encarga de los monstruos personalizados y de los spawners del servidor. Sus objetivos son:

- Registrar los monstruos creados en la configuración.
- Evitar que los monstruos originales del juego aparezcan.
- Cargar los monstruos creados en la configuración.

- Cargar los spawners creados en la configuración.
- Controlar las entidades del servidor.
- Ofrecer herramientas a los administradores para la creación de spawners.
- Capturar los eventos relacionados con los monstruos y cargas/descargas del mapa para un control eficiente de la memoria del servidor.

Las clases que intervienen en este módulo son: (Se citarán las importantes y se agruparán las que constituyan grandes grupos con el mismo fin).

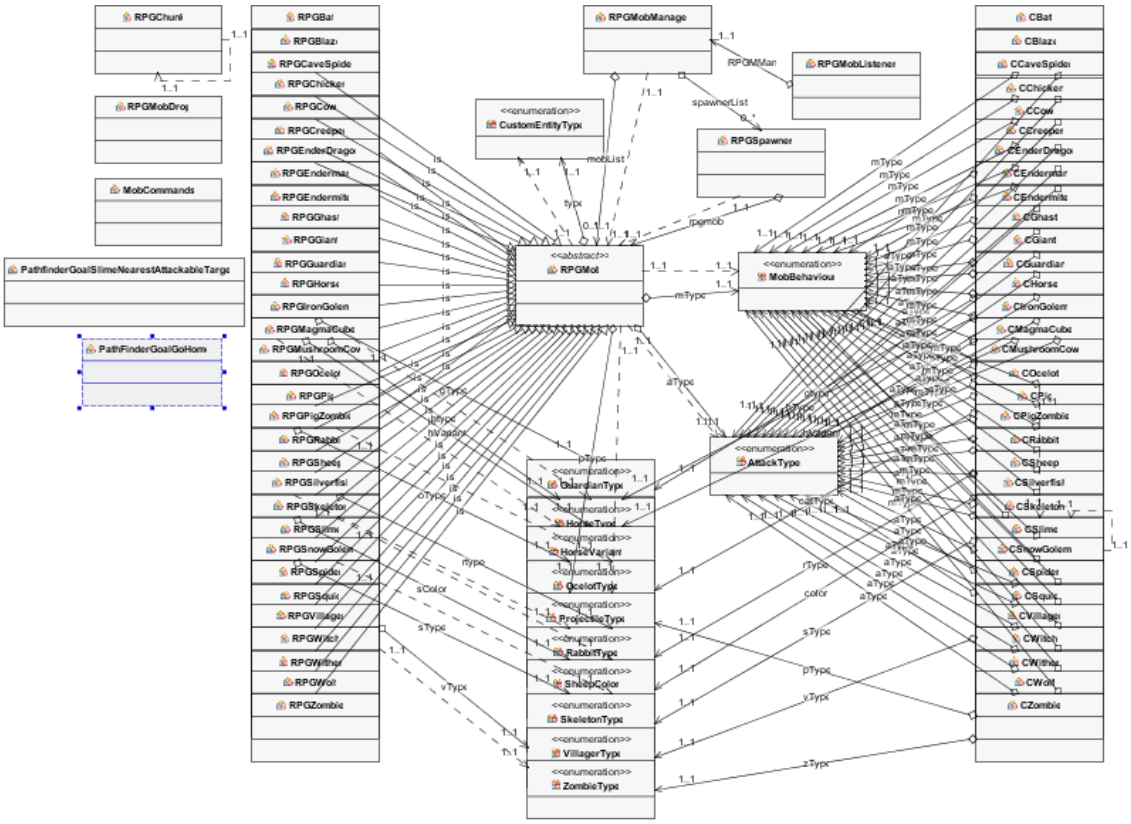
- ***RPGMobManager***: Clase que gestiona la carga de las configuraciones de monstruos y de los spawners del servidor.
- ***RPGMobListener***: Clase que captura los eventos producidos por los monstruos y las cargas/descargas del mapa, para una gestión eficiente de los recursos del servidor, con tal de habilitar/deshabilitar los spawners. Además captura eventos producidos normalmente por el servidor como la aparición de monstruos originales del juego.
- ***RPGMobCommands***: Clase que ofrece herramientas a los administradores para la creación de spawners e invocación de monstruos.
- ***RPGMob***: Clase modelo de carácter abstracto que representa cualquier tipo de monstruo, contiene los atributos básicos que contiene cualquier criatura, entre ellos: Nivel, nombre, tipo de monstruo, tipo de comportamiento, tipo de ataque, atributos propios de los monstruos (daño físico, velocidad de movimiento, velocidad de ataque, radio de persecución, daño a distancia, etcétera), dinero y experiencia.
- ***CustomEntityType***: Enumerado que contiene los tipos de monstruos posibles en Minecraft. Contiene:
    - Herramientas necesarias para reescribir los mapas internos del servidor y registrar las nuevas entidades.
    - Herramientas para la invocación de un monstruo en un punto de un punto.
    - Herramientas de reflexión para la modificación de atributos de los monstruos.
- Clases de nombre RPG + nombre de monstruo: Clase que extiende a la clase ***RPGMob***. Cada clase representa a un monstruo distinto y solo contiene los datos referentes a ese tipo de monstruo. En el caso del ***RPGBat***, que es un murciélago básicamente decorativo en el juego, contiene además los atributos propios de la clase Bat de Minecraft, en este caso solo contiene uno, “asleep” que indica si el murciélago debe aparecer aleteando o recogido. Si se visualiza ***RPGHorse*** se encontrarán además tipos de caballos, y variantes de colores de caballos.
- Clases de nombre C + nombre de monstruo: Clases de los monstruos que contienen los constructores y herramientas para sobre-escribir el comportamiento y atributos originales de los monstruos originales de Minecraft. Hacen uso de las clases de nombre RPG seguido de nombre de monstruo citada en el punto anterior para configurar sus atributos y comportamientos. Estas clases han sido descompiladas y modificadas con tal de añadir atributos y comportamientos que los mobs hostiles tenían para compartirlos con otras entidades de carácter pacífico.

- ***RPGMobDrop***: Clase que representa el objeto que deja caer un monstruo al morir. Contienen los siguientes datos: Tipo de objeto, el objeto y la probabilidad de que caiga al suelo.
- ***RPGChunk***: Clase que representa la localización de un *chunk*[34]. Un *chunk* es una porción de mapa y es necesario tener acceso directo a cada uno de ellos, sin depender de toda la estructura de un *chunk,* para gestionar de manera eficiente que spawners están activos/desactivados.
- ***PathFinderGoalGoHome*** y ***PathFinderGoalSlimeNearestAttackableTarget***: Son clases que modelan el comportamiento de un monstruo, son necesarios para el control de los monstruos en el servidor. ***PathFinderGoalGoHome*** es común a todos los monstruos creados, evita que al perder su objetivo (haya muerto o alejado de su rango de persecución) deambule por el mapa y vuelva al lugar donde apareció. El segundo *pathfinder*[35] se ha generado con tal de evitar que el monstruo *Slime* pierda su objetivo como en el juego original.
- ***MobBehaviour***: Enumerado que lista los distintos comportamientos que un monstruo puede tener, entre ellos: pacífico (no ataca), neutral (no ataca mientras no reciba daño), hostil (ataca cuando se está en su radio de acción) y dos más que son modificaciones de las dos últimas, ambas tienen la misma función, cuando atacan si más monstruos del mismo tipo están cerca vendrán en su ayuda.
- ***AttackType***: Enumerado que lista los distintos tipos de ataques que un monstruo puede realizar, en esta versión se encontrarán: Ataque físico (no ataca hasta que la distancia es muy corta del objetivo) y ataque a distancia (lanzará flechas en el momento en el que su objetivo esté en su radio de acción).
- El resto de enumerados son aspectos visuales que un mismo monstruo puede tener, por ejemplo el monstruo Horse (caballo) dispone de tipos y variantes, los tipos son: caballo, burro, asno, zombi y esqueleto; y las variantes son los distintos tipos de pelajes que pueden tener.

El diagrama de clases de este módulo es el siguiente, por limpieza los atributos y métodos han sido ocultados.

[34] Chunk: porción de mapa de 16x16x256bloques (ancho, largo y alto, respectivamente)

[35] Pathfinder: nombre que recibe los diferentes tipos de inteligencia artificial aplicada a los monstruos de Minecraft.



6.2.1.13 Módulo de configuración global

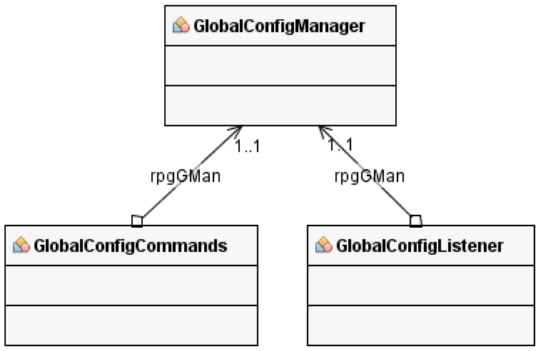
Este módulo se encarga de cargar la configuración global del servidor. Sus objetivos son:

- Cargar la configuración global del servidor.
- Ofrecer herramientas a los administradores para gestionar la configuración.
- Capturar eventos que tienen relación con el módulo de configuración global.

Las clases involucradas en este módulo son:

- ***GlobalConfigManager***: Clase que se encarga de cargar la configuración al iniciar al servidor.
- ***GlobalConfigCommands***: Clase que ofrece herramientas a los administradores para manejar la configuración global.
- ***GlobalConfigListener***: Captura los eventos que tienen relación directa con la configuración, entre ellos: ***BlockPlaceEvent***, ***BlockDestroyEvent***, ***FoodLevelChangeEvent***, ***PlayerJoinEvent***, ***EntityDamageByEntity***, etcétera.

El diagrama de clases de este módulo es el siguiente, por limpieza los atributos y métodos han sido ocultados.



6.2.2 Diseño de la aplicación de escritorio

Los módulos de la aplicación de escritorio gestora de las configuraciones del complemento irán acompañados por una descripción, la información que manejan, diagramas de clase explicativos y diagramas de secuencia en el caso de que sea necesario. Cada módulo tendrá un gestor, y en el caso que sea necesario clases que representen distintos tipos de objetos que maneje el complemento en ejecución.

Para el diseño de la aplicación se ha utilizado Swing en el entorno de desarrollo Netbeans mediante la creación de un formulario que contiene las pestañas con las configuraciones de los distintos módulos que se ponen a disposición del administrador del servidor. Los módulos serán separados por pestañas que serán las siguientes:

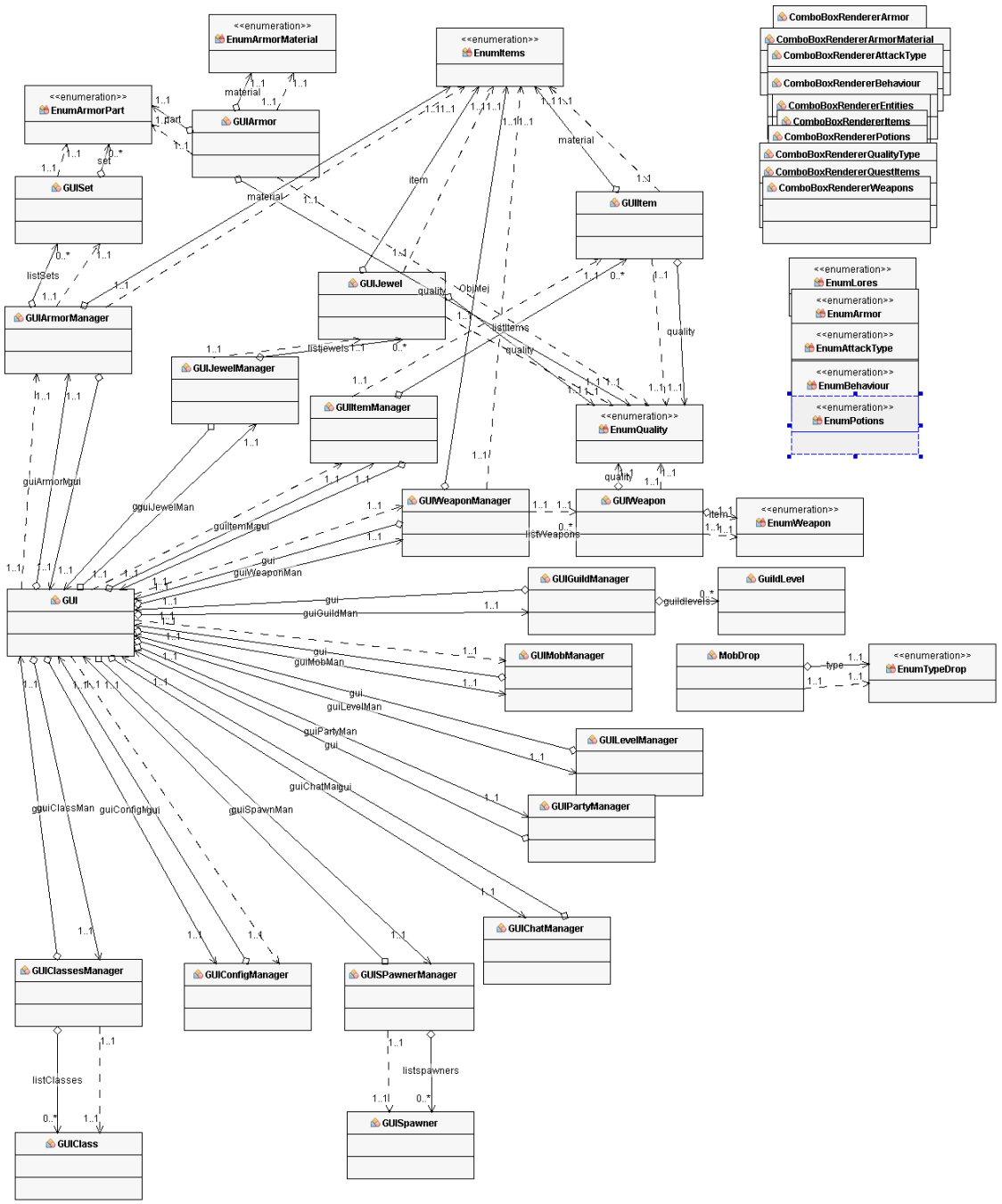
- General: Encargada de las configuraciones generales del complemento.
- Chat: Se ocupa de las configuraciones de los distintos canales de comunicación en el servidor.
- Niveles: Encargada de las configuraciones de la progresión del nivel de los jugadores.
- Clases: Se ocupa de gestionar la configuración de las clases o profesiones que los jugadores podrán elegir.
- Clanes: Encargada de la gestión de la configuración de los clanes.
- Grupos: Se responsabiliza de la configuración de los grupos en el complemento.
- Objetos: Se ocupa de la creación de objetos para el juego.
- Joyas: Se encarga de la creación de joyas para el juego y de la configuración para su mejora.
- Armas: Crea las armas del juego y la configuración para su mejora.
- Armaduras: Crea sets de armaduras y la configuración para la mejora de las partes que los componen.
- Monstruos: Encargada de crear monstruos en el juego y poder configurar atributos, comportamiento y los objetos que dejarán caer al morir
- Spawners o generadores de monstruos: Encargados de crear los puntos donde los monstruos de la pestaña anterior aparecerán.

Al igual que se mencionó en el diseño del complemento, los módulos que componen esta aplicación de escritorio están acoplados, ya que varios de los módulos por los atributos de configuración que contienen están relacionados con otros. Estás pestañas representan en si los distintos módulos de los que consta esta parte del proyecto.

Cada módulo de configuración modificará los ficheros de configuración propios, en algunos casos serán 1 o 2 ficheros, a continuación una equivalencia de módulo y ficheros de configuración:

- Módulo de configuración general:
  - globalConfig.yml.
- Módulo de configuración de niveles:
  - levelsConfig.yml.
- Módulo de configuración de clases:
  - Clases.yml.
- Módulo de configuración de clanes:
  - guildConfig.yml.
- Módulo de configuración de grupos:
  - partyConfig.yml.
- Módulo de configuración de objetos:
  - questItems.yml.
- Módulo de configuración de joyas:
  - jewelConfig.yml: Contiene la configuración del módulo de joyas.
  - jewels.yml: Contiene las joyas.
- Módulo de configuración de armas:
  - weaponConfig.yml: Contiene la configuración del módulo de armas.
  - weapons.yml: Contiene las armas.
- Módulo de configuración de armaduras:
  - armorConfig.yml: Contiene la configuración del módulo de armaduras.
  - armor.yml: Contiene las armaduras
- Módulo de configuración de monstruos:
  - mobs.yml: Contiene los monstruos.
- Módulo de configuración de generadores de monstruos:
  - spawners.yml: Contiene los generadores de monstruos.

Debido al tamaño total de la parte de la aplicación de escritorio, es imposible mostrar todo el diagrama de clases por lo que se han ocultado métodos y atributos de las clases que la forman.



Una característica notable de la interfaz son los selectores de monstruos, objetos, armaduras, joyas y armas que tendrán una representación visual del objeto acompañando al nombre del mismo, ayudando al administrador a crear una configuración.

Otra característica añadida es la posibilidad de ver en una gráfica la pendiente de los niveles de los jugadores y visualizar los datos nivel y experiencia en una tabla.

Los objetivos de la aplicación de escritorio son:

- Aportar herramientas a los administradores para la creación y modificación de la configuración de los módulos del complemento.

- Crear una configuración inicial, por defecto al iniciar la aplicación se crean las carpetas y ficheros Yaml de configuración del complemento. Si estas ya existieran se cargarían las configuraciones que contuvieran.
- Mejorar el sistema de configuración que, comúnmente, traen los complementos para servidores no oficiales de Minecraft, ofreciendo una visión más interactiva al administrador, evitando modificar ficheros Yaml a mano usando esta herramienta.

A continuación se seguirá con los módulos con una breve descripción, objetivos, clases que lo componen, prototipo y que datos que manejan si es necesario.

6.2.2.1 Módulo de configuración global

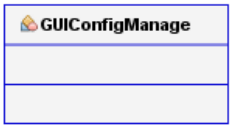
En este módulo se puede configurar aspectos sobre el daño que reciben los jugadores propios de Minecraft y se pueden habilitar o deshabilitar otros módulos. Los objetivos de este módulo son:

- Crear una configuración por defecto, en caso de que no existiera una antes.
- Permitir modificar al administrador los aspectos generales del juego.
- Crear una configuración modificada por el administrador.

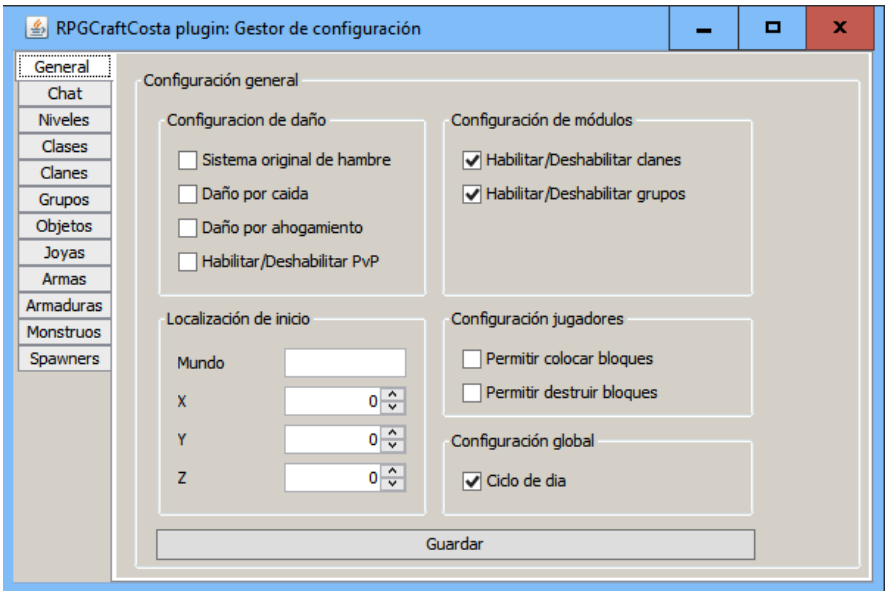
Las clases que intervienen en este módulo son:

- ***GUIConfigManager***: Esta clase es la encargada de gestionar la configuración general del complemento, carga los datos del fichero y los pasa a la interfaz, y de igual manera cuando se quiere guardar la nueva configuración recibe los datos y los almacena en el fichero Yaml.

El diagrama de clases consta de una sola clase que es la propia gestora de la configuración general del complemento. Por limpieza los atributos y métodos de la clase han sido ocultados.



A continuación un prototipo de la pestaña del módulo de configuración general.



Como se puede observar en esta pestaña se pueden gestionar de qué maneras los jugadores recibirán daño modificando el comportamiento normal del juego en Minecraft. Además se pueden habilitar o deshabilitar los módulos de clanes y grupos a conveniencia del administrador, crear un punto en el que reaparecerán los jugadores una vez se conecten o mueran y gestionar otros permisos de jugador como los de poder modificar el mapa.

6.2.2.2 Módulo de configuración de niveles

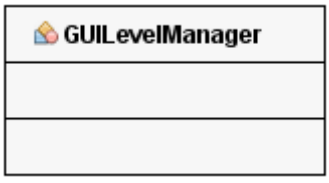
Este módulo es el encargado de la configuración de los niveles que los jugadores podrán alcanzar acumulando experiencia. Los objetivos de este módulo son:

- Crear una configuración por defecto en el caso de que no exista.
- Cargar la configuración de niveles.
- Pre- visualizar una gráfica y una tabla con las equivalencias de nivel–experiencia requerida.
- Verificar que la configuración es correcta.
- Guardar la configuración nueva.

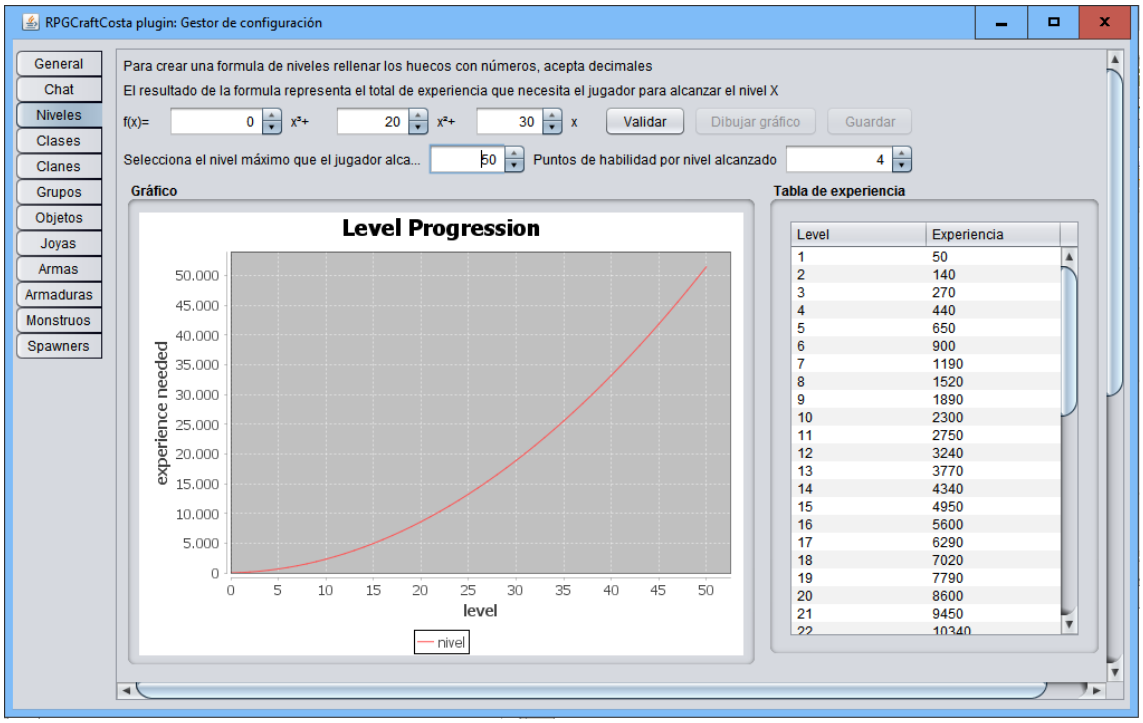
La clase que compone este módulo es:

- ***GUILevelManager***: Esta clase gestiona la configuración de niveles cargando y modificando la configuración que contienen los ficheros *Yaml*.

El diagrama de clases del módulo de configuración de niveles solo consta de una clase. Por limpieza los atributos y métodos de la clase han sido ocultados.



A continuación un prototipo de la pestaña de la configuración de niveles.



En la pestaña de niveles se puede configurar una fórmula para que los jugadores suban de nivel en el caso de que alcancen cierta experiencia y limitar el nivel máximo del jugador a uno concreto. Una vez que la fórmula este validada se podrá dibujar una gráfica que representará la pendiente de dificultad del juego, además de una tabla que mostrará los pares nivel-experiencia para visualizar cuanta experiencia es necesaria para alcanzar cierto nivel.

6.2.2.3 Módulo de configuración de clases

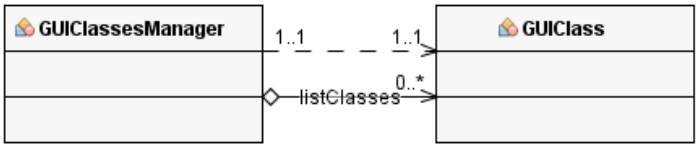
Este módulo es el encargado de gestionar las clases o profesiones de los jugadores que a posteriori serán usadas por el complemento. Sus objetivos son:

- Cargar las clases de los ficheros de configuración.
- Permitir crear, modificar y eliminar clases o profesiones.
- Validar los datos de las clases creadas o modificadas.

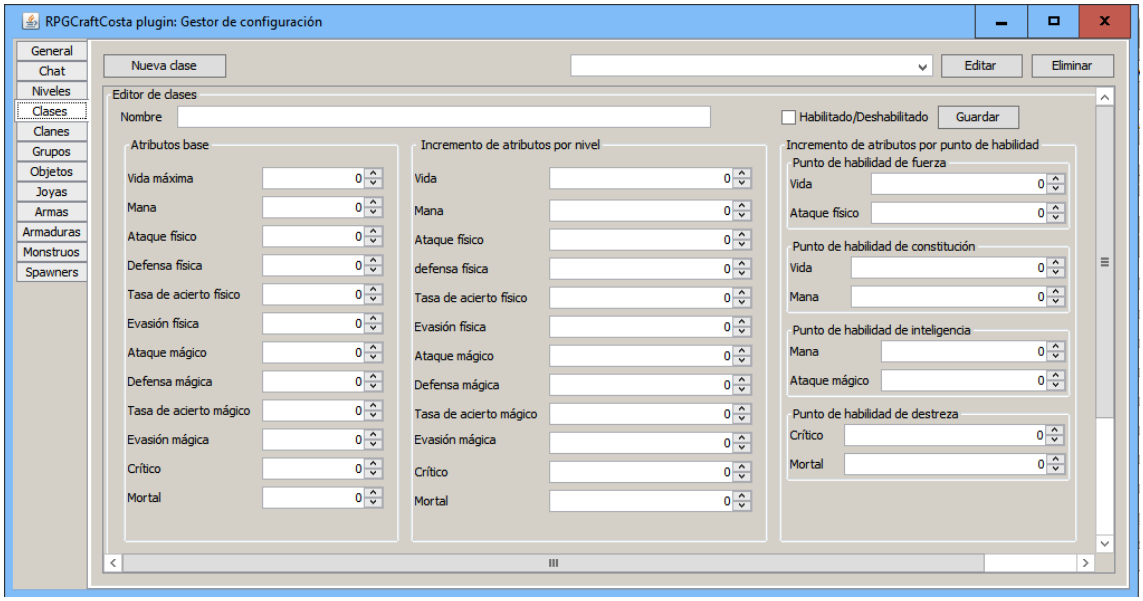
Las clases que intervienen en este módulo son:

- ***GUIClassesManager***: Clase encargada de la carga, guardado y validación de las clases que se creen, modifiquen o eliminen de los ficheros de configuración de clases.
- ***GUIClass***: Clase modelo que representa una clase o profesión. Contiene los atributos físicos, mágicos que obtendrá el jugador al acceder a la clase, además de las mejoras al obtener nuevos niveles y aplicar los puntos de habilidad.

El diagrama de clases del módulo de configuración de clases es el siguiente.



A continuación un prototipo de la pestaña de clases.



En esta pestaña se podrá crear, editar y eliminar las clases o profesiones que los jugadores podrán elegir al conectarse por primera vez al servidor. En tres columnas principales se encuentran: los atributos base de la clase, las mejoras sobre los atributos que los jugadores adquieren al subir de nivel y las mejoras de los atributos al repartir los puntos de habilidad que el jugador adquiere al subir de nivel.

6.2.2.4 Módulo de configuración de chats

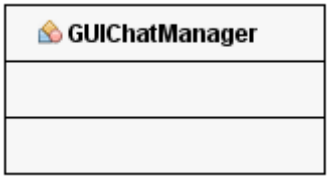
Este módulo es el encargado de gestionar la carga y guardado de la configuración de los canales de comunicación del complemento. Los objetivos de este módulo son:

- Crear una configuración por defecto para los canales de comunicación.
- Permitir modificar los aspectos como prefijos, los colores de los mensajes y atajos de teclado para su uso en el servidor.
- Validar la configuración antes de guardar la configuración nueva.
- Guardar la nueva configuración.

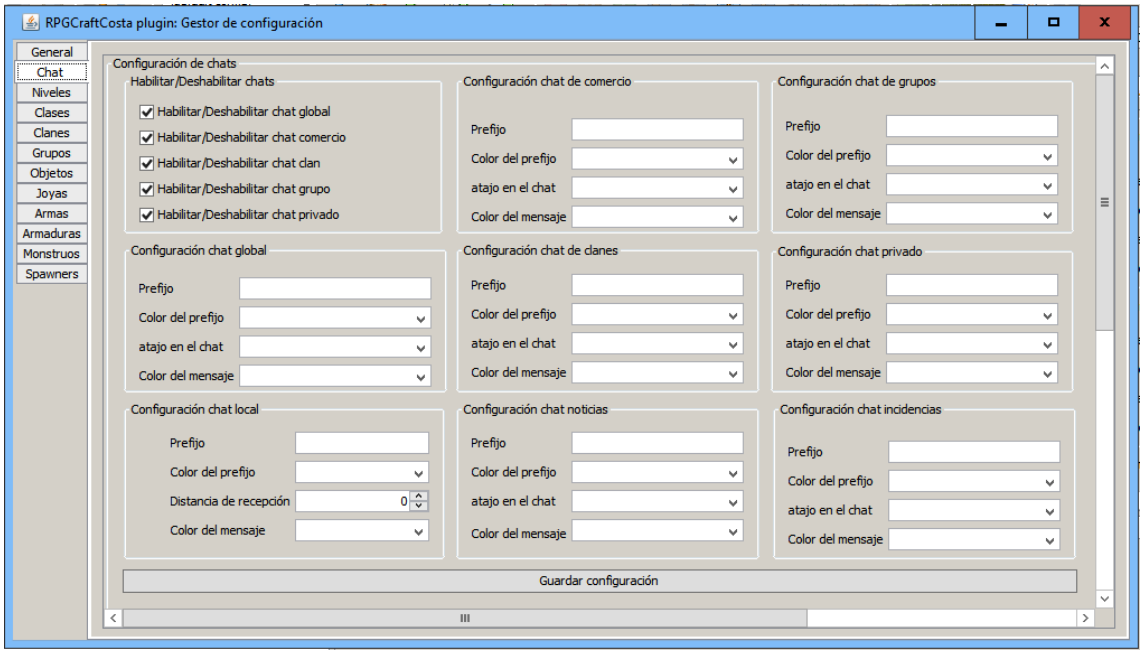
La clase que interviene en este módulo es:

- ***GUIChatManager***: Esta clase es la encargada de la carga, validación y guardado de la configuración de los canales de comunicación del complemento.

El diagrama de clases del módulo de configuración de chat es el siguiente:



A continuación un prototipo de la pestaña de chats.



En la interfaz de configuración de chats se pueden ver nueve paneles diferenciados, el primero habilita o deshabilita los chats que los jugadores pueden usar, los ocho siguientes gestionan como los chats serán representados en el juego, de tal manera que se podrá elegir un prefijo, un atajo de chat y los colores del prefijo y del mensaje para diferenciarlos de otros canales de comunicación.

6.2.2.5 Módulo de configuración de clanes

Este módulo se encarga de la configuración de los clanes, sus objetivos son:

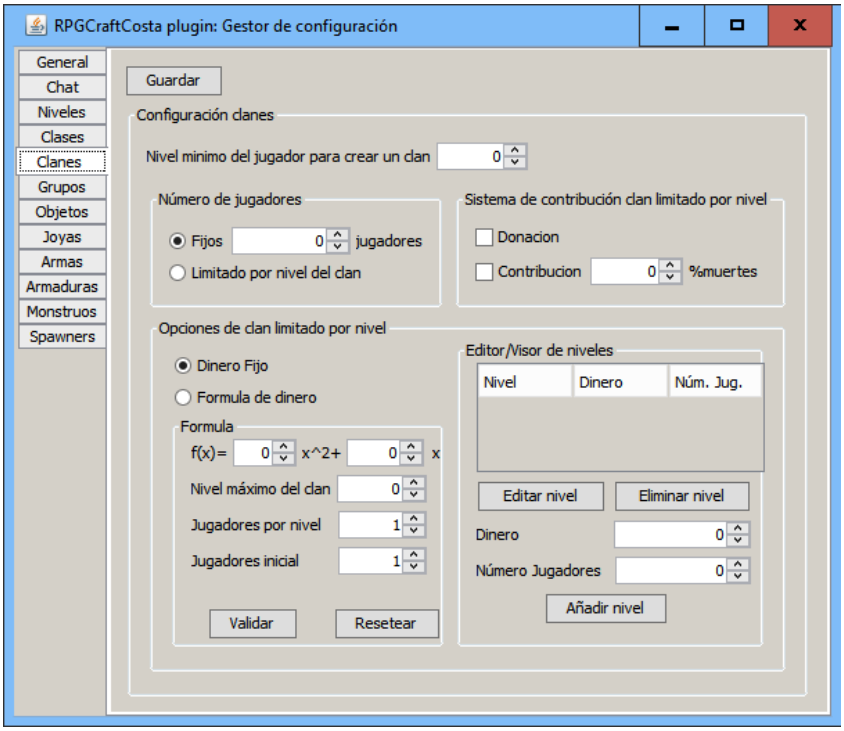
- Crear una configuración por defecto.
- Cargar la configuración existente
- Permitir modificar la configuración de los clanes, modificando atributos como número de jugadores, nivel mínimo para crear un clan, limitar el número de jugadores por nivel del clan y ofrecer herramientas para la mejora del nivel del clan, entre otras opciones.
- Validación de la configuración creada o modificada.
- Guardar la nueva configuración.

Las clases que componen este módulo son:

- ***GUIGuildManager***: Es la clase encargada de la carga, modificación, validación y guardado de la configuración de los clanes. Contiene los atributos de configuración de los clanes como: número máximo de jugadores, nivel mínimo para crear un clan, formula de creación de niveles y sistema de contribución al nivel del clan, ya sea por donación o por contribución mediante un impuesto sobre el dinero que los monstruos dan al jugador cuando son matados.
- ***GuildLevel***: Si en la configuración se elige el sistema de clanes basados en nivel esta clase crea las instancias de los niveles que los clanes pueden alcanzar.

El diagrama de clases de este módulo es el siguiente:

Y el prototipo de la pestaña de configuración de clanes es el siguiente:



Desde la pestaña de configuración de clanes se puede configurar, los jugadores que puede albergar un clan están ilimitados, fijos o limitados por el nivel del clan, el nivel mínimo para crear o unirse a un clan. Si el número de jugadores del clan está limitado por nivel, se podrán crear los niveles del clan manualmente o usar una fórmula que los creará automáticamente hasta cierto nivel al que se limite. Además si están limitados por nivel, para alcanzar nuevos niveles se podrá seleccionar las maneras en las que el clan adquiere esos niveles, que podrán ser mediante donaciones directamente desde el jugador o indirectamente como un porcentaje del dinero que los jugadores reciben de los monstruos que matan.

6.2.2.6 Módulo de configuración de grupos

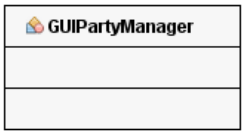
Este módulo se encarga de la configuración de grupos, sus objetivos son:

- Crear una configuración por defecto.
- Cargar la configuración existente.
- Permitir modificar la configuración de grupos, modificando atributos como: número de jugadores del grupo, sistema de reparto de dinero y experiencia entre el grupo formado por los jugadores.
- Guardar la configuración de grupos.

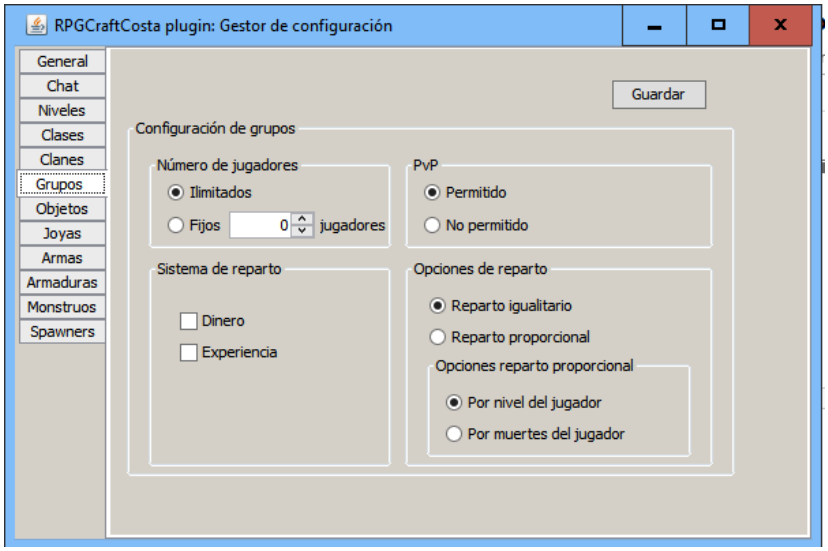
La clase de la que consta este módulo es:

- ***GUIPartyManager***: Se encarga de la carga, modificación y guardado de la configuración de grupos.

El diagrama de clases de este módulo es el siguiente:



El prototipo de la pestaña de configuración de grupos es el siguiente:



En la interfaz de configuración de grupos, al igual que en los clanes se podrá limitar el número de jugadores. Lo interesante de formar grupos es que permiten compartir las ganancias de experiencia y dinero a sus componentes, por lo que se podrá elegir que ganancias y de qué manera estas son repartidas.

6.2.2.7 Módulo de configuración de objetos

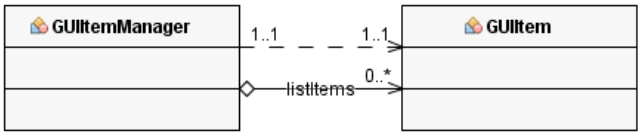
Este módulo se encarga de la configuración de los objetos en el complemento, sus objetivos son:

- Crear algunos objetos por defecto.
- Cargar los objetos ya creados por la configuración.
- Listar los objetos creados.
- Crear, editar y eliminar objetos.
- Permite modificar el nombre, objeto y las líneas de descripción del objeto que son mostrados en la interfaz del juego.

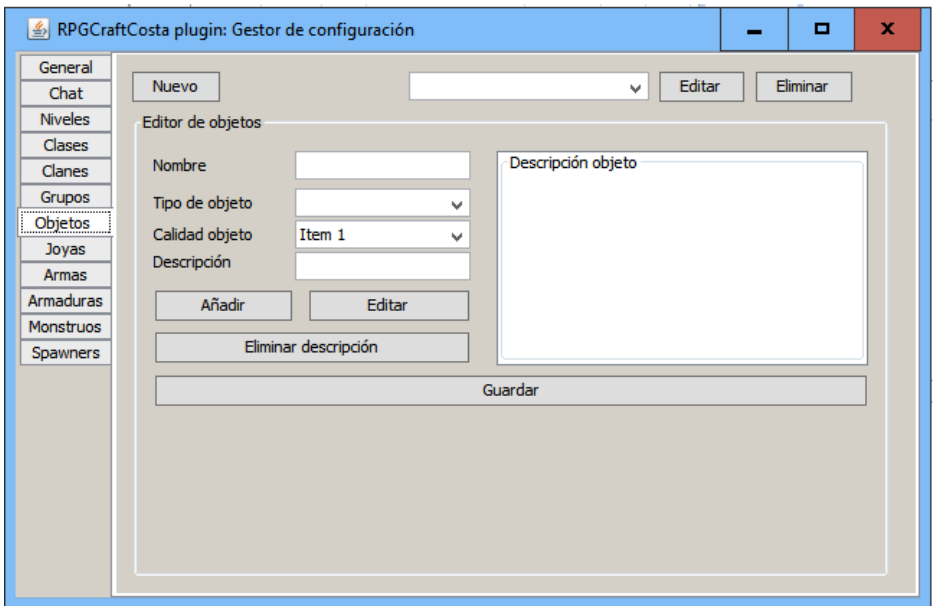
Las clases que componen este módulo de configuración son:

- ***GUIItemManager***: Clase encargada de la carga, listado, modificación y eliminación de los objetos del fichero de configuración de objetos.
- ***GUIItem***: Clase modelo de un objeto que contiene un nombre, el objeto al que se corresponde en Minecraft y una descripción.

El diagrama de clases de este módulo de configuración es el siguiente:



El prototipo de la pestaña de configuración de objetos es la siguiente:



En esta interfaz de configuración de objetos se podrá renombrar y añadir descripciones a los objetos originales del videojuego, que se comportarán de la misma manera que en el juego original. Por ejemplo si en la configuración global se tiene marcada la opción “Sistema de hambre original” se deberán añadir objetos comestibles para que los jugadores sacien su hambre y evitar morir de inanición.

6.2.2.8 Módulo de configuración de joyas

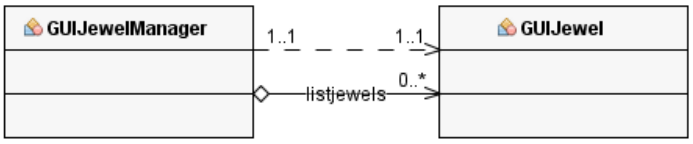
Este módulo se encarga tanto de la configuración de joyas como de su creación, sus objetivos son:

- Crear una configuración y algunas joyas por defecto.
- Permitir modificar la configuración de las joyas, ya que son objetos mejorables en el complemento mediante la combinación de joyas.
- Listar las joyas existentes en configuración.
- Permitir la creación, edición y eliminación de joyas del complemento.
- Validar y guardar la configuración de joyas.

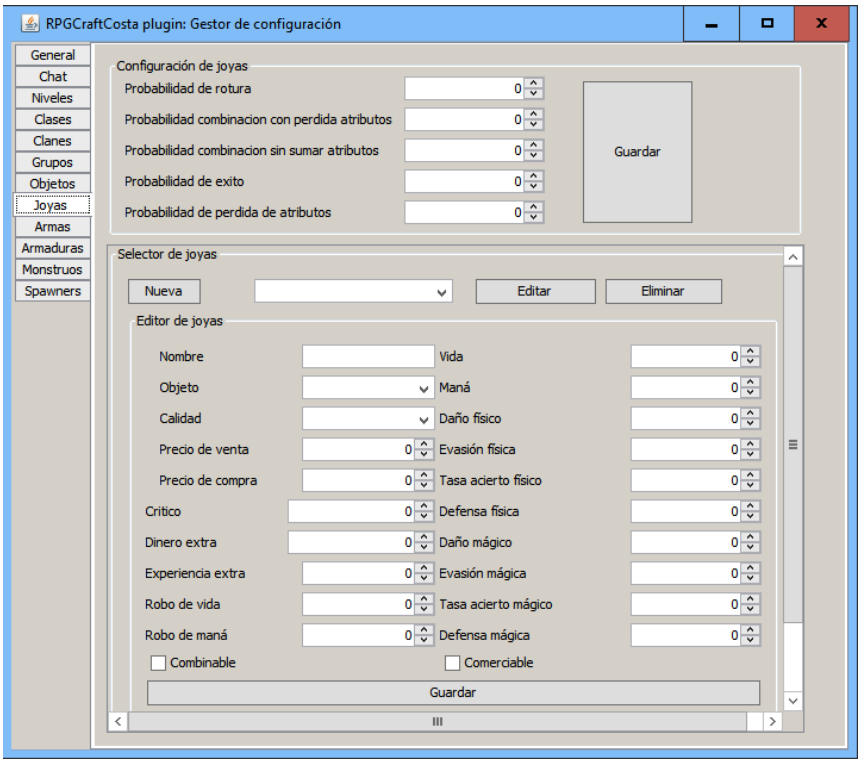
Las clases involucradas en este módulo son:

- ***GUIJewelManager***: Es la clase encargada de la gestión de la configuración y de las joyas existentes en los ficheros de configuración.
- ***GUIJewel***: Clase modelo de una joya que contiene los atributos de una joya, entre ellos: Nombre, objeto que lo representa en el juego y una serie de atributos que mejoran los del jugador que las porte equipadas.

El diagrama de clases de este módulo es el siguiente:



El prototipo de la pestaña de configuración de joyas es el siguiente:



En esta pestaña de configuración de joyas se podrá configurar el rango de porcentajes de éxito o fracaso de combinación de joyas, de esta manera se obtiene una mayor aleatoriedad en este tipo de objetos. Por otra parte, se podrá crear, editar y eliminar las joyas en la configuración del complemento.

6.2.2.9 Módulo de configuración de armas

Este módulo se encarga de la configuración de armas y de la creación de armas en el complemento, sus objetivos son:

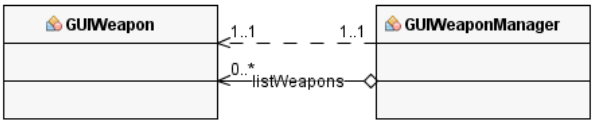
- Crear una configuración de las armas por defecto y algunas armas.
- Listar las armas existentes en la configuración.
- Permitir modificar la configuración de las armas, que consiste en crear un objeto para la mejora de armas.
- Permitir la creación edición y eliminación de armas.
- Permitir guardar la configuración de armas.

Las clases que intervienen en este módulo son:

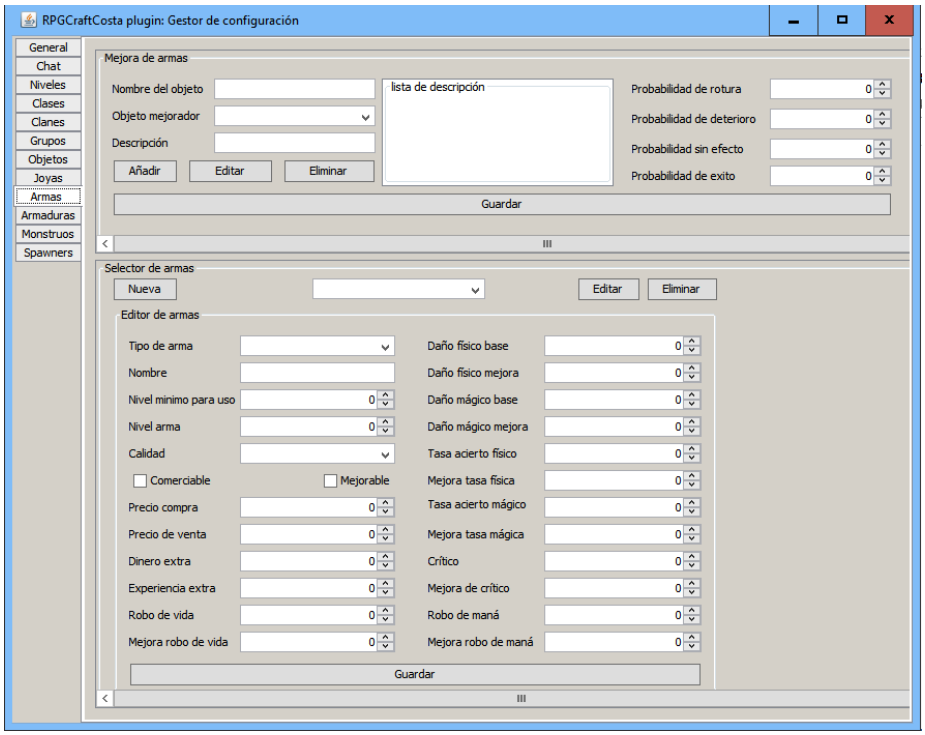
- ***GUIWeaponManager***: Clase encargada de la gestión de la configuración y de las armas cargadas desde los ficheros de configuración del complemento.

- ***GUIWeapon***: Clase modelo de un arma que contiene los atributos de un arma, entre ellos: Nombre, nivel requerido para su uso, calidad del arma, atributos de daño y tasas de acierto, entre otros.

El diagrama de clases de este módulo es el siguiente:



El prototipo de la pestaña de configuración de las armas es el siguiente:



Al igual que las joyas las armas también se pueden mejorar, pero por un procedimiento distinto, el uso de objetos mejoradores. Estos objetos se aplican sobre el arma y se aplican los rangos de probabilidad para obtener un éxito, deterioro o rotura del arma. Más abajo se encontrará el selector de armas donde se podrán crear, editar y eliminar las armas, asignándoles atributos propios como atributos de daño y de tasas de acierto entre otras opciones.

6.2.2.10 Módulo de configuración de armaduras

Este módulo se encarga de la configuración y de las armaduras del complemento, sus objetivos son:

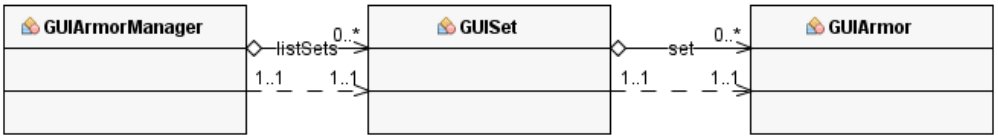
- Crear una configuración por defecto y algunas armaduras.
- Cargar la configuración y las armaduras que se encuentran en los ficheros de configuración.
- Listar las armaduras y mostrar la configuración de las armaduras.
- Permitir la creación de un objeto que mejore las armaduras.
- Permitir la creación, edición y eliminación de sets de armaduras completos.

- Guardar la configuración de armaduras.

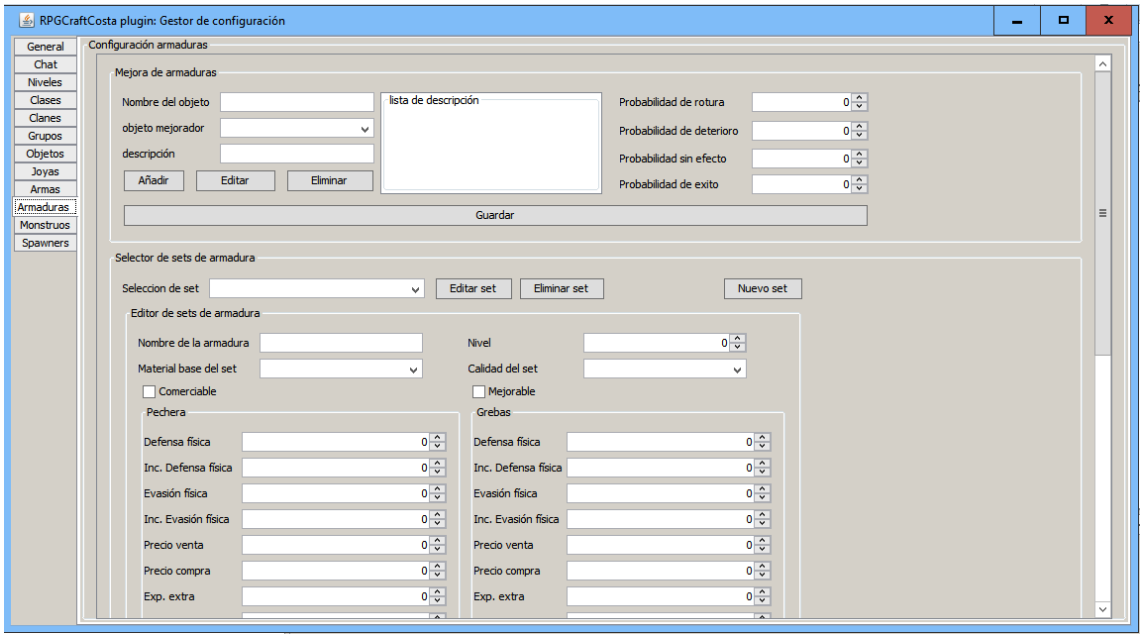
Las clases que componen este módulo de configuración son:

- ***GUIArmorManager***: Esta clase se encarga de la gestión de la configuración y de las armaduras del complemento.
- ***GUISet***: Clase modelo que representa un set de armadura completo, constituido de cuatro partes (casco, pechera, grebas y botas).
- ***GUIArmor***: Clase modelo que representa una parte de un set de una armadura, entre sus atributos se encuentran: nombre de la pieza, material del que está elaborado, atributos de carácter defensivo, entre otros.

El diagrama de clases de este módulo de configuración es el siguiente:



El prototipo de la pestaña de configuración de armaduras es el siguiente:



En la pestaña de configuración de armaduras al igual que en el de configuración de armas se encuentra el apartado de mejora de partes de armadura, en el que se podrá crear un objeto mejorador. En el selector de armaduras de la pestaña de configuración se podrá crear, editar o eliminar un set de armadura, asignando atributos de carácter defensivo como defensa física y tasa de evasión de golpes a cada parte de la armadura por separado.

6.2.2.11 Módulo de configuración de monstruos

Este módulo se encarga de la creación de los monstruos del complemento, sus objetivos son:

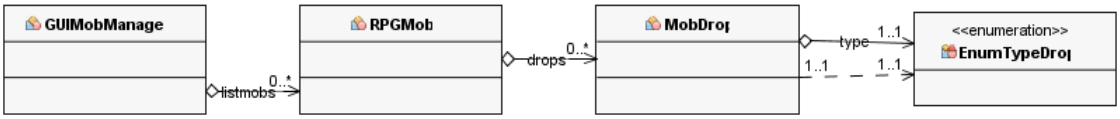
- Crear unos monstruos por defecto.

- Listar los monstruos existentes en la configuración.
- Permitir crear, editar y eliminar los monstruos.
- Gestionar el dinero, experiencia y objetos que un monstruo deja al morir.

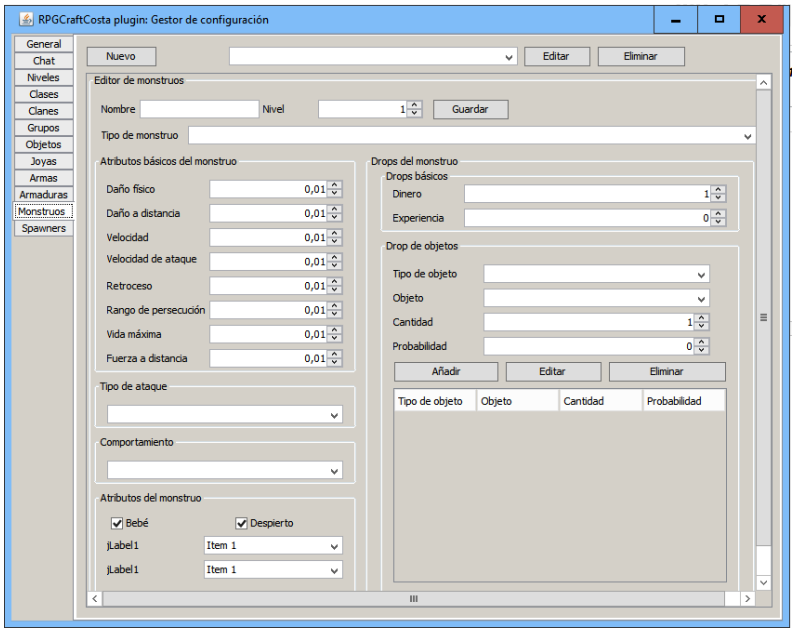
Las clases que componen este módulo de configuración son:

- ***GUIMobManager***: Clase encargada de la gestión de monstruos en el complemento.
- ***RPGMob***: Clase modelo que representa un monstruo en el complemento y contiene los datos y atributos que definen un monstruo. Esta clase es compartida por el complemento y por la aplicación de escritorio.
- ***RPGMobDrop***: Clase modelo de un objeto que es dejado caer por un monstruo al morir, contiene datos que personalizan el comportamiento de estos objetos, como son la cantidad de ellos que puede dejar caer un monstruo o la probabilidad de que caigan.
- ***EnumTypeDrop***: Clase enumerada que lista los tipos de objetos creados por la aplicación de configuración y que facilitan la selección del tipo de objetos que un monstruo deja caer al morir, entre ellos. Mejoradores de armadura o armas, joyas, partes de armadura, objetos y armas.

El diagrama de clases de este módulo de configuración es el siguiente:



El prototipo de la pestaña de configuración de monstruos es el siguiente:



En la pestaña de configuración de monstruos se podrán crear, editar o eliminar monstruos de nuestro complemento. Se podrá asignar el tipo de monstruo, el tipo de ataque, el daño, el comportamiento de los monstruos (pacíficos, neutrales o agresivos) y el dinero, experiencia y objetos que el jugador podrá recibir una vez sea matado. Además ciertos monstruos del juego de Minecraft pueden tener distintas apariencias,

como es el caso del caballo, ocelote, zombi y otros tantos, por lo que se podrán personalizar.

6.2.2.12 Módulo de configuración de generadores

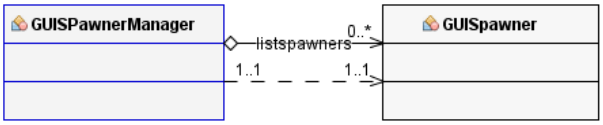
Este módulo se encarga de la configuración de los generadores de monstruos del complemento, sus objetivos son:

- Cargar la configuración que haya sido creada (Por defecto no se crea ningún generador, dado que estos se ubican en puntos exactos en el mapa, si esos puntos están bloqueados por bloques el monstruo queda atrapado y muere a causa de los bloques).
- Permitir la creación, edición y eliminación de los generadores que posteriormente usará el complemento.

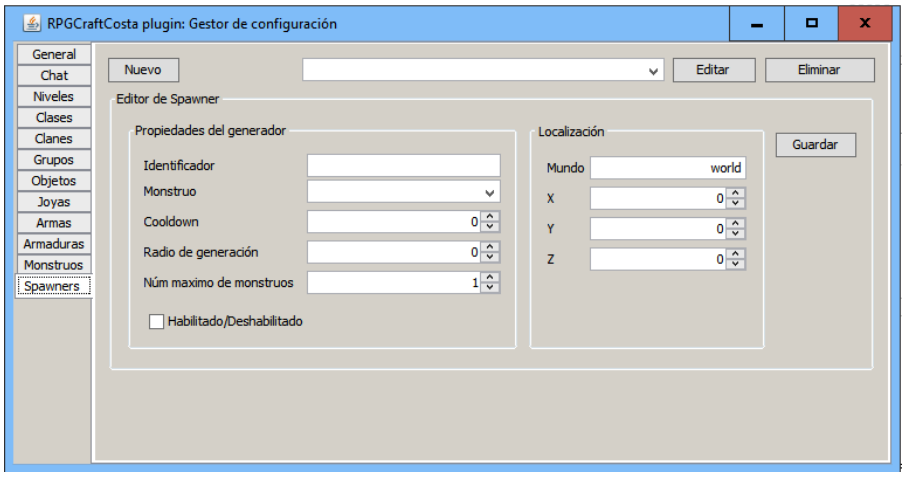
En este módulo intervienen las siguientes clases:

- ***GUISpawnerManager***: Clase gestora de la configuración de los generadores de monstruos.
- ***GUISpawner***: Clase modelo que representa un generador de monstruos. Entre la información que contiene se encuentra la localización en coordenadas del generador y el mundo en el que se encuentra, el tipo de monstruo que debe aparecer, la cantidad de monstruos que aparecen y cada cuanto tiempo son reemplazados los que son muertos.

El diagrama de clases de este módulo de configuración es el siguiente:



El prototipo de la pestaña de configuración de generadores es el siguiente:



La pestaña de configuración de generadores de monstruos crea, edita y elimina los spawners del complemento. En el apartado llamado “Monstruo” aparecerán los monstruos que se hayan creado hasta el momento para poder ser seleccionados.

7 Implementación

En este apartado se hará hincapié en los aspectos relevantes del proyecto que ofrezcan una mejora sobre los que ya existen en el entorno de desarrollo de complementos para servidores no oficiales de Minecraft.

7.1 Interfaz de configuración

Los complementos para servidores no oficiales de Minecraft, de manera general, ofrecen la posibilidad de configurar el complemento tras la instalación del mismo, ya que una vez instalados es cuando aparecen los ficheros de configuración en la carpeta “plugins” del servidor, ralentizando la tarea. Mediante la creación de esta interfaz es posible crear una configuración antes de la instalación, con la única condición de transportar esa configuración al servidor antes de instalar el complemento.

La aplicación de escritorio crea, en su misma carpeta, la jerarquía de ficheros de configuración necesarios para que el complemento instale y cargue la configuración que se haya creado por defecto de una sola vez, evitando los pasos como la configuración tras la instalación del complemento y los posibles apagados o encendidos del servidor para aplicar las nuevas configuraciones.

El posible defecto que se puede encontrar a este método de creación de configuración, es el alto grado de conciencia que requiere el administrador para crear una nueva configuración relacionada con los aspectos generales que un juego de rol contiene, como pueden ser: los niveles de dificultad del juego, que atributos o comportamientos deben tener los monstruos, la facilidad de obtención de objetos, experiencia y dinero, y equilibrar todo lo anterior para crear una dinámica del juego que no aborrezca al jugador.

7.2 Monstruos

Los monstruos originales de Minecraft están limitados a los comportamientos, atributos y a los niveles de dificultad que el juego tiene por defecto. Es un punto débil del juego difícil de superar, ya que limitan las acciones que un servidor puede realizar para la modificación de los monstruos. Cambiar la naturaleza de un animal o monstruo en Minecraft requiere de un cambio de su posición en la jerarquía dentro del código fuente de Minecraft, lo cual es una tarea titánica para alguien que apenas ha tenido contacto con la reflexión de java, ni trabajado con código ofuscado. Se ha tenido que descompilar el servidor de Minecraft no oficial, trabajado con código ofuscado y usado la reflexión que ofrece Java para mediar entre los monstruos originales y los creados. Se explicara de qué manera se ha conseguido crear nuevas entidades de las que ya existían.

Los monstruos de Minecraft (animales y monstruos) siguen una jerarquía dentro de los paquetes del servidor de Minecraft y extienden de paquetes más generales que los definen de un tipo o de otro. Sin ir más lejos los caballos, ocelotes y lobos pueden ser amaestrados, mientras que los demás no lo son, pero esto es solo un ejemplo pequeño. Existen muchos tipos de monstruos, pueden ser: pacíficos, hostiles, neutrales, de los que atacan a distancias largas o mediante ataques físicos. La intención de este proyecto es aunarlos todas estas características en un mismo tipo de manera que se puedan crear de un mismo monstruo una infinidad de posibilidades. Para ello ha sido necesario estudiar la jerarquía de los monstruos en el código fuente del servidor de Minecraft para hacer que extiendan de una clase que no les prive de los atributos que contienen y

principalmente poder hacer que cualquier entidad ataque o cambie su comportamiento según sea configurada. Para ello han sido necesarios los siguientes pasos:

- Reescribir las 32 clases de los monstruos y animales que existen en Minecraft 1.8 usando de referencia las clases descompiladas de los monstruos originales. Modificar la clase de la que extienden a ***EntityMonster*** (la cual permite a los animales y monstruos atacar), implementar la interfaz ***IRangedEntity*** (para permitir a cualquier monstruo o animal atacar a distancia) y añadir los constructores que permiten modificar no solo el comportamiento y las formas de ataque sino los atributos internos y de apariencia del monstruo o animal.
- Crear los medios para el registro de nuevas entidades en el servidor de Minecraft mediante el uso de reflexión para poder modificar los mapas de entidades registradas normalmente en el servidor. En el código del servidor se encuentra una clase denominada ***EntityType*** la cual es la encargada de registrar todas las entidades en los mapas de entidades del servidor, en este caso ha sido necesario crear una nueva clase llamada ***CustomEntityType*** de tipo enumerado que añade cada uno de los 32 monstruos y animales que he creado a los que ya existen en Minecraft. Permitiendo así su creación en el juego.
- Para trabajar con los ficheros de configuración de monstruos y crear la fábrica de monstruos para el complemento ha sido necesario crear una jerarquía paralela que facilite la creación y cargado de las nuevas entidades, separando lo que es código puramente del servidor de lo que es el código del complemento.
- Creación de *pathfinder* adaptado para los monstruos que simule el comportamiento normal de un monstruo cuando es alejado de su punto de creación y no tiene objetivo actual, que consiste en devolverlo a su origen.
- El comportamiento y modos de ataque de un monstruo o animal viene marcado por los *pathfinders* que le son asignados, son en sí mismo la inteligencia artificial de los monstruos. Para ello se han creado 5 tipos de comportamientos y 2 de ataque que modelan los distintos tipos de comportamientos que se pueden encontrar en un juego de rol básico, se explicarán a continuación:
    - Comportamiento pacífico: Este comportamiento típico de un animal original de Minecraft se caracteriza por pasear cerca de donde apareció y de huir del jugador si es atacado.
    - Comportamiento neutral: Este comportamiento se caracteriza por que solo atacara en defensa y seguirá a su objetivo si este no acaba huyendo.
    - Comportamiento hostil: Se caracteriza por que si un posible oponente aparece en su radio de acción este monstruo lo perseguirá e intentará atacar a la mínima posibilidad que se le presente.
    - Comportamiento neutral con llamada a compañeros: Es igual que el comportamiento neutral salvo que si el monstruo está rodeado de monstruos de su mismo tipo ayudarán a atacar al objetivo del monstruo que fue atacado.
    - Comportamiento hostil con llamada a compañeros: Es igual que el comportamiento hostil salvo que si el monstruo está rodeado de monstruos de su mismo tipo ayudarán a atacar al objetivo del monstruo.
    - Ataque físico: Este tipo de ataque requiere que la distancia entre el monstruo y su oponente sea muy corta, por lo que el monstruo deberá acercarse al objetivo antes de atacar.

    - Ataque a distancia: Este tipo de ataque no requiere de una distancia corta pero si de una mínima distancia para que el monstruo ataque al oponente. En esta versión los ataques a distancia son flechas lanzadas por los monstruos, aunque se pueden personalizar el tipo de ataque a distancia con el lanzamiento de otros objetos.
- Modificar los métodos que definen el aspecto visual de un monstruo para que se puedan seleccionar directamente desde el constructor, mediante la modificación de los metadatos de las entidades.
- Modificar el sistema de objetos que dejan caer al morir los monstruos, para ello se debe modificar las clases de los monstruos para eliminar los objetos que originalmente dejaban caer y mediante la aplicación de escritorio asignar los tipos de objetos creados en la configuración.

7.3 Generadores de monstruos

Un elemento fundamental de todo RPG son los puntos de generación de monstruos, por lo general estos puntos almacenan un solo tipo de monstruos y pueden aparecer 1 o varios cada cierto periodo de tiempo. Para crear esta funcionalidad ha sido necesario investigar en profundidad como es el cargado/descargado del mapa en el servidor y como las entidades aparecen en Minecraft. Para ello han sido necesarios los siguientes pasos:

- Adquisición de conocimientos sobre:
    - *Chunks*: Los *chunks* (porción en español) son trozos de mapa de 16x16x256 (ancho, largo y alto) que almacenan la información sobre las entidades y bloques que contienen. Estos no son constantes puesto que son cargados y descargados cuando el jugador está próximo. Los monstruos que aparecen aleatoriamente en estas porciones de mapa son invocados dependiendo del bioma o entorno en el que se encuentren (pantano, isla, montaña, valle, etcétera), tipo de mundo (*World* o mundo normal, *Nether*[36] o inframundo, *The end*[37] o El fin) o franja horaria en la que se encuentren (día o noche).
    - Generadores de monstruos: Los generadores de monstruos o *spawners* en Minecraft, son las herramientas del juego original para la aparición de monstruos de un mismo tipo en una localización exacta. A pesar de que su objetivo es muy similar al perseguido por este proyecto, no permiten la asignación de entidades que no son las originales del juego por lo que han sido desechados como una posibilidad y obligan a crear nuestros propios generadores de monstruos.
- Crear los medios para la gestión de los generadores de monstruos, pudiendo cargar o descargar estos dependiendo de, respectivamente, las cargas y descargas del mapa, control de las entidades creadas en todo momento, evitar los posibles sobrecostes de búsquedas de entidades mediante estructuras de datos afines.

[36] Nether: inframundo que representa el infierno por sus característicos lagos de lava y cuevas.

[37] The end: El fin representa en el juego individual la fase final del videojuego, un mundo paralelo accesible solo desde un portal y el lugar donde se encuentra el jefazo “Dragón del fin”.

- Captura de eventos referentes a la aparición de los monstruos (***CreatureSpawnEvent***) con tal de limitarla y permitir solo los creados por el complemento.
- Captura de los eventos referentes a la carga y descarga del mapa en el servidor (***ChunkLoadEvent*** y ***ChunkUnloadEvent***) con tal de gestionar los generadores que en ellos se encuentren.
- Creación de los hilos que gestionarán las entidades que cada generador contiene. Con cada carga y descarga estos hilos serán creados y destruidos, con tal de mantener el uso de memoria bajo y la coherencia de las estructuras de datos de las entidades en el juego.

8 Resultados

En este apartado se mostrarán los resultados obtenidos tras el desarrollo de este proyecto.

Las pruebas de integración del complemento y de la aplicación han sido totalmente satisfactorias puesto que cumplen los requisitos definidos en los apartados anteriores, con un total de 51000 líneas de código aproximadamente la funcionalidad del proyecto está terminada.

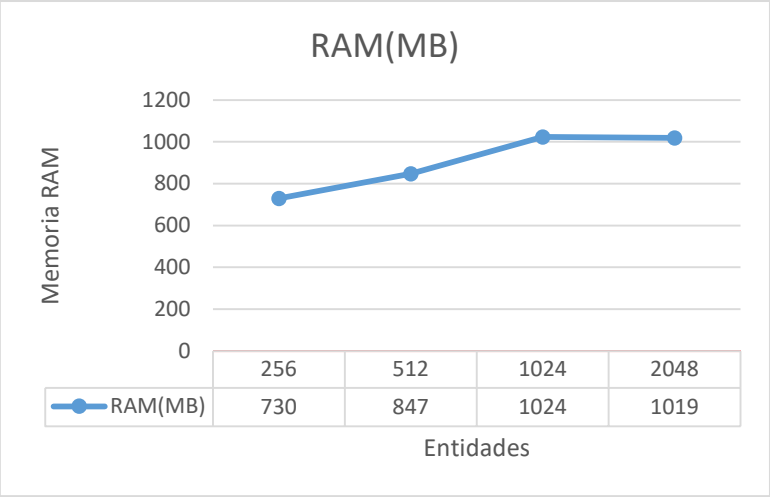
A continuación se mostrarán los resultados de las pruebas de rendimiento del complemento sobre un entorno de producción afín, es decir, un servidor hospedado en Internet con las siguientes características:

- Memoria RAM ilimitada
- Procesador Xeon E5-1650v2
- Frecuencia base 3.5 Ghz
- Frecuencia turbo 3.9 Ghz
- 5 GB SSD
- Consola de servidor en vivo
- Estadísticas en tiempo real CPU/RAM/Jugadores

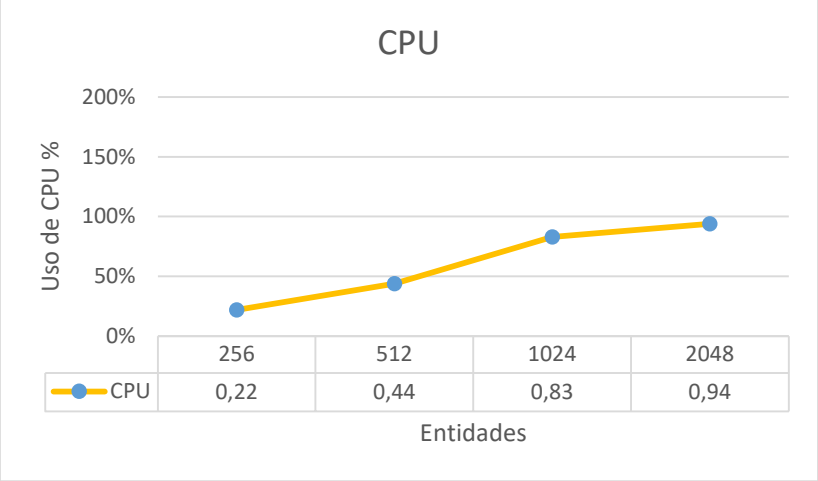
Las pruebas de rendimiento consistirán en la creación incremental de generadores de monstruos, puesto que son los que mayor repercusión sobre el rendimiento del servidor pueden provocar, sin tener en cuenta las cargas de las porciones de mapa. Se analizarán los consumos de RAM, CPU, los TPS[38] del servidor y los procesos del complemento que mayor repercusión tienen en el rendimiento del servidor mediante las herramientas que el propio servidor y el hosting ofrecen.

Para las pruebas de rendimiento se han creado 4 escenarios de prueba en los que se incrementaran el número de generadores y de entidades en el servidor. Cada generador administra un hilo de ejecución que hace aparecer un monstruo en su localización, cuando este muere sea la causa que sea reaparecerá otro manteniendo estable el escenario. Los 4 escenarios consisten en crear 256, 512, 1024 y 2048 generadores en una misma zona cargada en la memoria del servidor para ver la influencia sobre la RAM, CPU, TPS del servidor. A continuación los resultados:

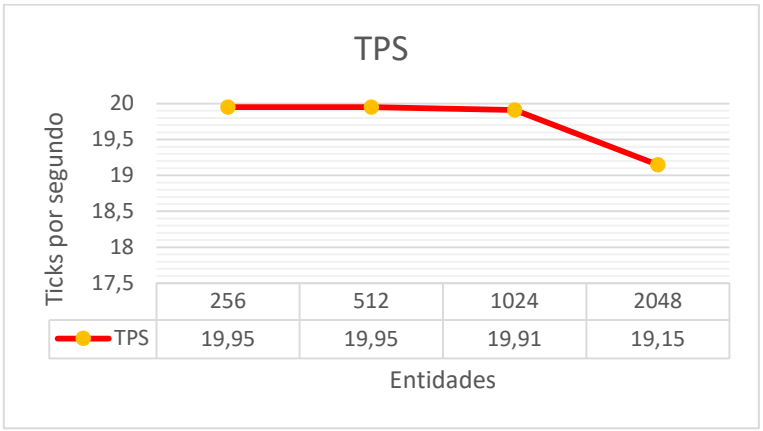
[38] TPS: Ticks por segundo es la velocidad a la que la CPU del servidor responde a las tareas, el nivel máximo que puede alcanzar es de 20.



Como se observa en este gráfico el aumento de consumo de memoria RAM queda patente al incrementar el número de generadores activos en el servidor. El incremento de entidades no es directamente proporcional a la memoria ubicada.



En cuanto al uso de CPU, se observa que al duplicar la cantidad de entidades el uso de CPU se duplica. En la gráfica se tuvo en cuenta los procesadores con los que contaba el servidor, en este caso 2 procesadores y por ello el máximo uso de CPU se considera un 200%.



Los ticks por segundo o TPS representan la velocidad a la que el CPU gestiona el estado actual del servidor. De manera general, 20 Ticks por segundo indican que la CPU no está sobrecargada de trabajo y es la cota superior de esta medida. Cuando esta medida baja indica que la CPU no da abasto con el trabajo que tiene entre manos y requiere tiempo de otros recursos creando retrasos, más conocido como lag, que los jugadores sufren en la parte cliente. Como se observa en el grafico los TPS siguen una distribución lineal descendente poco pronunciada y dentro de los valores aceptables, una medida inferior a 17 TPS podría considerarse inaceptable para el desarrollo normal del videojuego.

En la ejecución de los análisis mediante las herramientas que ofrece el servidor Spigot se han obtenido que clases, objetos o eventos influyan negativamente al rendimiento del servidor, entre ellos:

- ***RPGSpawnerBukkitRunnable***: Clase que crea los hilos que gestionan las entidades de cada generador de monstruos, afectando en un 1.75% de los TPS del servidor.
- Las entidades creadas por el complemento afectan en un 1.28% de los TPS del servidor.
- La inteligencia artificial de las entidades creadas afectan en un 1.39% de los TPS del servidor.

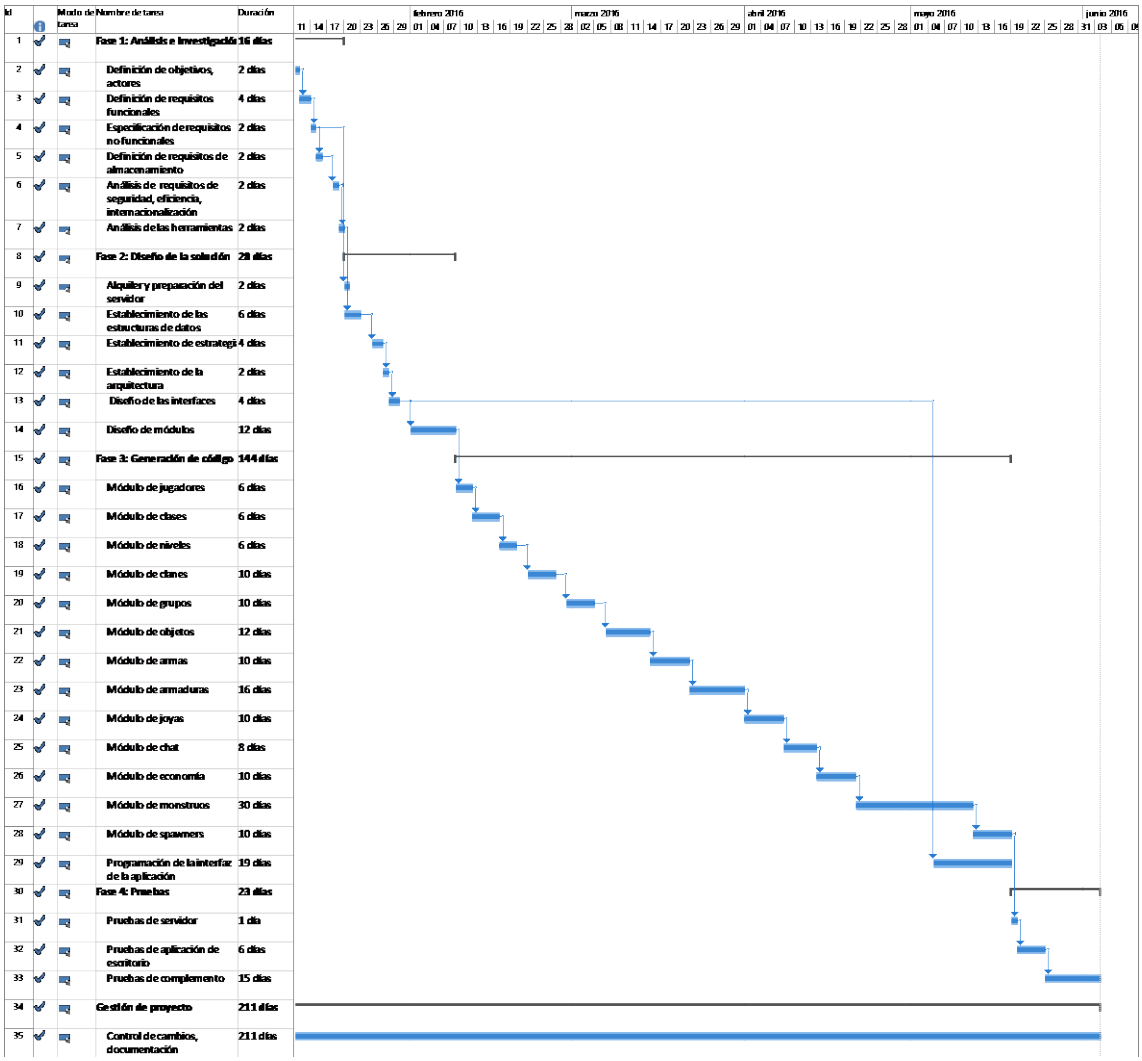
Estos resultados indican que el impacto de estas nuevas entidades creadas y los generadores son un potencial cuello de botella si no son bien administrados en el servidor. El complemento está preparado para este tipo de imprevistos, como ya se avanzó en el apartado de implementación, ya que si un generador no está en uso porque una zona del mapa no está cargada en memoria destruye el hilo y las entidades. Las pruebas realizadas sobre el rendimiento son casos excepcionales que difícilmente pueden ocurrir en un escenario real dado que los generadores suelen distanciarse unos de otros en el mapa y no todos actúan al mismo tiempo, pero puede darse el caso de que cientos de jugadores repartidos en distintas zonas del mapa creen este cuello de botella.

8.1 Análisis de los resultados

En este apartado se analizarán los resultados sobre la solución propuesta y los obstáculos que ralentizaron el proceso de desarrollo de este proyecto.

La solución propuesta ha sido desarrollada satisfactoriamente en términos de funcionalidad pero no en término de los tiempos que se estimaron en un primer presupuesto, debido a obstáculos tecnológicos, desconocimiento del paradigma de programación dirigido por eventos y la ingeniería inversa aplicada a los códigos ofuscados descompilados del servidor no oficial Spigot. Estos obstáculos tuvieron su mayor impacto en las fases de generación de código y pruebas, y más concretamente en los módulos de monstruos.

A continuación se ha elaborado un nuevo presupuesto con las dificultades encontradas, a partir del siguiente diagrama de Gantt.

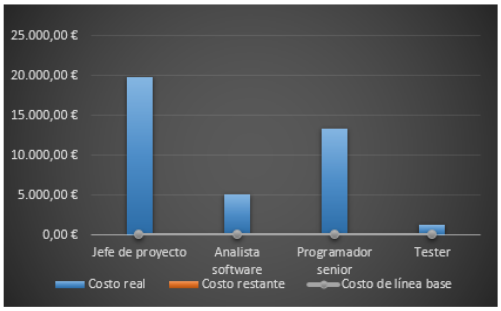


Id	Modo de tarea	Nombre de tarea	Duración
1		**Fase 1: Análisis e Investigación**	**16 días**
2		Definición de objetivos, actores	2 días
3		Definición de requisitos funcionales	4 días
4		Especificación de requisitos no funcionales	2 días
5		Definición de requisitos de almacenamiento	2 días
6		Análisis de requisitos de seguridad, eficiencia, internacionalización	2 días
7		Análisis de las herramientas	2 días
8		**Fase 2: Diseño de la solución**	**28 días**
9		Alquiler y preparación del servidor	2 días
10		Establecimiento de las estructuras de datos	6 días
11		Establecimiento de estrategi	4 días
12		Establecimiento de la arquitectura	2 días
13		Diseño de las interfaces	4 días
14		Diseño de módulos	12 días
15		**Fase 3: Generación de código**	**144 días**
16		Módulo de jugadores	6 días
17		Módulo de clases	6 días
18		Módulo de niveles	6 días
19		Módulo de clanes	10 días
20		Módulo de grupos	10 días
21		Módulo de objetos	12 días
22		Módulo de armas	10 días
23		Módulo de armaduras	16 días
24		Módulo de joyas	10 días
25		Módulo de chat	8 días
26		Módulo de economía	10 días
27		Módulo de monstruos	30 días
28		Módulo de spawners	10 días
29		Programación de la interfaz de la aplicación	19 días
30		**Fase 4: Pruebas**	**23 días**
31		Pruebas de servidor	1 día
32		Pruebas de aplicación de escritorio	6 días
33		Pruebas de complemento	15 días
34		**Gestión de proyecto**	**211 días**
35		Control de cambios, documentación	211 días

El desarrollo completo de este proyecto consta de 1576 horas divididos entre los distintos recursos humanos de los que se han hecho uso, presentando un aumento de 200 horas aproximadamente y un 13% de desviación de la planificación original.

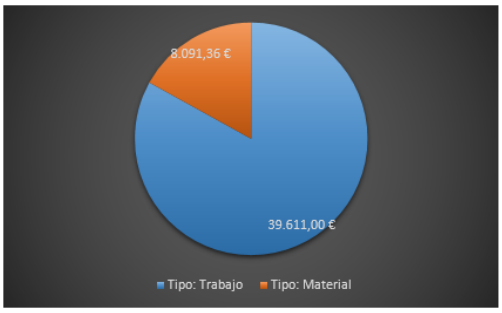
ESTADO DEL COSTO

Estado de costo de los recursos de trabajo.



DISTRIBUCIÓN DE COSTOS

Cómo los costos están distribuidos entre tipos de recursos diferentes.



DETALLES DE COSTOS

Detalles de costos de todos los recursos de trabajo.

Nombre	Trabajo real	Costo real	Tasa estándar
Jefe de proyecto	660 horas	19.806,00 €	30,00 €/hora
Analista software	168 horas	5.150,00 €	30,00 €/hora
Programador senior	660 horas	13.320,00 €	20,00 €/hora
Tester	88 horas	1.335,00 €	15,00 €/hora

El presupuesto total del proyecto es de 47.702,36 €, lo que representa una desviación de un 15,6% del presupuesto estimado.

INFORMACIÓN GENERAL COSTOS

LUN 11/01/16 VIE 03/06/16

COSTO

47.702,36 €

COSTO RESTANTE

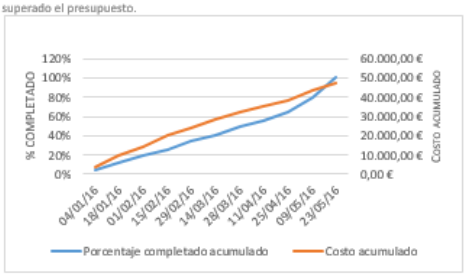
0,00 €

% COMPLETADO

100%

PROGRESO FRENTE A COSTO

Progreso realizado en comparación con el coste durante el proceso. Si el valor de la línea % completado está por debajo de la línea de coste acumulado, es posible que su proyecto haya superado el presupuesto.



ESTADO DEL COSTO

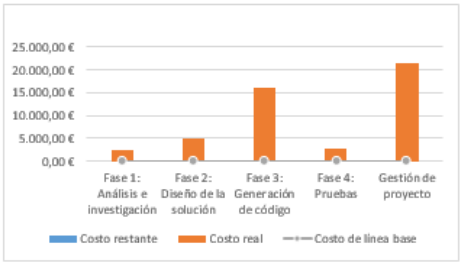
Estado de costo de tareas de nivel superior.

Nombre	Costo real	Costo restante	Costo de línea base	Costo	Variación de costo
Fase 1: Análisis e investigación	2.503,92 €	0,00 €	0,00 €	2.503,92 €	2.503,92 €
Fase 2: Diseño de la solución	4.849,76 €	0,00 €	0,00 €	4.849,76 €	4.849,76 €
Fase 3: Generación de código	16.179,84 €	0,00 €	0,00 €	16.179,84 €	16.179,84 €
Fase 4: Pruebas	2.848,92 €	0,00 €	0,00 €	2.848,92 €	2.848,92 €
Gestión de proyecto	21.319,92 €	0,00 €	0,00 €	21.319,92 €	21.319,92 €

ESTADO DE COSTO

Estado de costo de todas las tareas de nivel superior. ¿La línea base es cero?

Intente establecer una línea base



9 Conclusiones

Los objetivos que perseguía este proyecto se han visto cumplidos con la creación de un complemento MMORPG altamente configurable mediante una aplicación de escritorio que facilita en gran medida su configuración. La posibilidad de modificar la funcionalidad

de los distintos objetos, monstruos y dinámicas del juego no hace más que demostrar el alto nivel de flexibilidad de este videojuego.

A pesar de que es una versión muy reducida de un servidor MMORPG los posibles nuevos objetivos que serían consecuencia de este proyecto podrían ser la adición de habilidades especiales, efectos realizados mediante partículas, misiones para los jugadores, creación de una mayor amalgama de ataques para los monstruos, no solo limitándolos a ataques a corta y larga distancia sino a mágicos o que requieran de nuevos tipos de comportamientos, cerrando así el circulo de lo que un MMORPG contiene. Estos nuevos objetivos podrían ser desarrollados siguiendo el camino de esta solución adaptando el proyecto iterativamente a los nuevos módulos que estos crearían.

Para el desarrollo de este proyecto han sido necesarios conocimientos de muchos campos que se aprenden inicialmente en la carrera, de manera muy básica, y que han sido extendidos mediante la investigación, es el caso de: la reflexión de Java, estructuras de datos eficientes, hilos de ejecución, ingeniería inversa, patrones de diseño, repositorios, entornos de desarrollo integrado, administración de servidores, gestión de proyectos, aplicaciones FTP, inteligencia artificial, programación dirigida por eventos, entre otras tecnologías.

El principal problema que se encuentra en este tipo de desarrollos, es la envergadura que puede llegar a alcanzar acompañada de una la mala gestión que se puede realizar debido a la perdida de la visión global de los objetivos, por esa razón hay que hacer un mayor hincapié en las fases de análisis y diseño.

9.1 Relación trabajo con estudios cursados

Los campos de estudio que abarca este proyecto han sido vistos en la carrera de ingeniería del software, en algunos casos de manera muy básica como son: inteligencia artificial aplicada, reflexión, programación dirigida por eventos, repositorios y herramientas como entornos de desarrollo integrado; los cuales debido a la complejidad de este proyecto se ha tenido que realizar una investigación con tal de extender los conocimientos básicos aprendidos. En cambio, los hay que han sido vistos de una manera extensa, cosa que es de agradecer, como: patrones de diseño, gestión de proyectos, repositorios, hilos de ejecución, mantenibilidad del software, programación orientada a objetos y análisis de requisitos; los cuales han facilitado en gran parte a la culminación de este proyecto. Es de entender que no en todas las ramas de la carrera informática se aprenda todo al mismo nivel de detalle.

Los campos de estudio que no han sido vistos en la carrera de ingeniería del software o que simplemente consta de una mención, han requerido de una investigación por cuenta propia que ha retrasado en gran medida la culminación de este proyecto, como son: herramientas FTP, administración de servidores, la reflexión de Java, ingeniería inversa aplicada a software ofuscado entre otras.

Cabe destacar que sin los conocimientos adquiridos a través de los años de la carrera no hubiera sido posible la realización de este proyecto, no por el hecho de adquirir los conocimientos sino también por crear las bases para la comprensión y adquisición de nuevos conocimiento sobre tecnologías o metodologías que no se han visto en la carrera.

.

9.2 Trabajos futuros

Los posibles trabajos futuros consecuencia de este proyecto son:

- Implementar la internacionalización del complemento añadiendo los medios para los cambios de idioma tanto en los mensajes derivados del complemento como de la aplicación de escritorio de configuración.
- Implementación de una historia que guie a los jugadores a través del juego.
- Implementación de misiones: principales y secundarias repetibles para facilitar la adquisición de experiencia o bienes difíciles de encontrar de manera normal.
- Implementación de habilidades propias de las clases de personajes de rol, como pueden ser habilidades de restauración de vida de una clase de clérigo.
- Implementación de nuevos comportamientos y habilidades de los monstruos, no solo creando nuevos tipos de ataques sino añadiendo nuevos tipos de movimientos, por ejemplo la habilidad de volar.
- Implementación de efectos visuales que enriquezcan la apariencia del juego, dependientes de los eventos que se produzcan como: subir de nivel, efectos sobre ataques especiales o habilidades de personajes y monstruos. Se recomienda que para este tipo de implementación el envío de paquetes con partículas sea entre los jugadores más cercanos para evitar las sobrecargas de trabajo tanto en el servidor como de los clientes, puesto que un envío masivo de estos paquetes repercuten produciendo lag en los jugadores y una bajada del rendimiento del servidor.
- Implementación de personajes no jugadores que gestionen, misiones, adquisición de habilidades y comercio. Es posible mediante la fábrica de monstruos pero sería necesario crear nuevos comportamientos y capturar los eventos de los jugadores cuando interactúen con estos tipos de personajes.
- Implementación de un sistema de puntos de honor, por el cual se premia a los jugadores que realizan buenas prácticas, defendiendo al jugador de menor nivel en mundos de PvP[39].
- Implementación de servidores o mundos con distintas dificultades o atributos, como mundos sin PvP o dificultades incrementales. Es posible mediante el uso de la herramienta *BungeeCord* [40]es posible interconectar varios servidores para realizar el salto de los jugadores entre ellos sin necesidad de que el jugador vuelva a la interfaz del cliente para realizar la conexión.
- Mejora de los requisitos no funcionales del proyecto, entre ellos: rendimiento, escalabilidad, usabilidad, mantenibilidad y concurrencia. Intentando desacoplar gran parte de los módulos o gestionando de una mejor manera la memoria.
- Uso de base de datos para la carga y descarga de objetos y monstruos en el juego.
- Permitir cambiar la configuración en caliente sin perdidas de coherencia en el videojuego.

---

[39] PvP o jugador contra jugador es un modo de juego en el que los propios jugadores pueden luchar entre ellos.

[40] BungeeCord: herramienta de interconexión entre servidores https://www.spigotmc.org/wiki/about-bungeecord/

10 Bibliografía

11 [1]

12 «Minecraft», *Wikipedia, la enciclopedia libre*. 07-jul-2016.

13 [2]

14 D. Braun, *Let's Play Minecraft: Plugins programmieren mit Java*. MITP-Verlags GmbH & Co. KG, 2015.

15 [3]

16 J. W. Cooper, *Java Design Patterns: A Tutorial*. Addison-Wesley Professional, 2000.

17 [4]

18 S. R. Covey, *Los 7 hábitos de la gente altamente efectiva*. Grupo Planeta (GBS), 2014.

19 [5]

20 E. Gamma, R. Helm, R. Johnson, y J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader)*. Pearson Education, 1994.

21 [6]

22 M. T. Goodrich, R. Tamassia, y M. H. Goldwasser, *Data Structures and Algorithms in Java, 6th Edition*. Wiley Global Education, 2014.

23 [7]

24 A. Hunt, *Learn to Program with Minecraft Plugins: Create Flying Creepers and Flaming Cows in Java*, Edición: 1. Dallas, TX: Pragmatic Bookshelf, 2014.

25 [8]

26 P. M. Institute, *A Guide to the Project Management Body of Knowledge: PMBOK Guide*. Project Management Institute, 2013.

27 [9]

28 A. Kalinovsky, *Covert Java: Techniques for Decompiling, Patching, and Reverse Engineering*. Sams, 2004.

29 [10]

30 A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

31 [11]

32 J. T. P. Méndez, *Programación concurrente*. Editorial Paraninfo, 2003.

33 [12]

34 R. S. Pressman, *Ingeniería del software: un enfoque práctico*. McGraw-Hill, 1994.

35 [13]

36 C. Rogers, *Absolute Beginner's Guide to Minecraft Mods Programming*, Edición: 01. Indianapolis, Indiana: ALPHA BOOKS DIV OF PEARSON, 2014.

37 [14]

38 K. Rose, *Project Quality Management: Why, What and How*. J. Ross Publishing, Incorporated, 2014.

39 [15]

40 S. J. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

41 [16]

42 C. M. Sommer, *Building Minecraft Server Modifications*. Packt Publishing Ltd, 2015.

43 [17]
44 *Introducción a la Programación en Java.* ITM, 2007.
45 [18]
46 *Reverse Engineering of Design Patterns from Java Source Code*. ProQuest, 2007.

11 Anexos

1 Anexo I: Minecraft

1. Historia

Minecraft salió al mercado en su versión alfa para PC en mayo del 2009. Su creador, el desarrollador de videojuegos independientes y fundador de la empresa Mojang AB, Markus Alexej Persson, conocido como Notch, se inspiró en videojuegos como Infiniminer para desarrollar Minecraft, que originalmente recibió el nombre de Cave Game, un juego tipo «construcción libre» o SandBox.

En su versión Alfa el juego solo tenía dos modalidades: supervivencia, en el que el jugador deben adquirir recursos para prosperar; y creativo, en el que el jugador dispone de acceso libre a todos los materiales, no reciben daño y pueden volar, de manera que pueden realizar construcciones de cualquier tipo y extensión. Como anécdota de esta fase hubo un superfluo intento de una versión multijugador, que se hizo esperar hasta las pruebas de la versión Beta en paralelo a la alfa en Junio de 2009, en Septiembre del mismo año una caída del servidor de autentificación de los jugadores hizo que cualquier jugador, hubiera comprado o no el juego, pudiera disfrutar de la versión multijugador gratuitamente, cosa que Notch aprovechó con tal de dar a conocer Minecraft al mundo entero. Esta versión recibía actualizaciones semanales con tal de solucionar errores que los propios jugadores posteaban en un blog dedicado a ello. En diciembre de 2010 finalizó su desarrollo para dar paso a la versión Beta.

Su versión Beta no se hizo esperar ya que en diciembre de 2010 fue lanzada con tal de mejorar y pulir la jugabilidad, añadir nuevos contenidos y estabilidad al juego en general. Los jugadores que compraron la versión Alfa del videojuego tuvieron suerte ya que esta nueva versión aumento su precio considerablemente y no sufrieron recargo alguno. Esta versión aporto nuevos: enemigos (mobs), bloques, climas (biomas), PNJ (personaje no jugador) y una mejora visual de los elementos del juego. Esta fase de desarrollo terminó en septiembre de 2011 dando paso a su versión completa en noviembre del mismo año.

La revolución de Minecraft no se hizo esperar para llegar a otras plataformas ya que en Agosto de 2011 se lanzó la versión Pocket Edition para Android pero restringido a teléfonos inteligentes de la marca Sony por un acuerdo de exclusividad con Mojang, en Octubre del mismo año se lanzó una versión compatible con todos los teléfonos inteligentes tras finalizar el acuerdo. Para la plataforma iOS se retrasó el lanzamiento hasta noviembre del mismo año y actualmente apenas hay diferencias entre las versiones de esta plataforma y la de Android.

Con la llegada de la versión completa (Minecraft 1.0) también vino un sobrecargo en el precio del videojuego, como era de esperar los jugadores de las versiones anteriores no sufrió el recargo adicional. Esta versión añadía nuevas características: mejoras en cuanto a balance de daños, nuevos biomas y mundos, mejoras en ganadería, fortalezas, un jefe final, nuevos modos de creación de mundos y el modo extremo que añadía una dificultad significativa a la jugabilidad del videojuego.

Minecraft se lanzó para Xbox 360 en mayo de 2012, denominando a la versión Minecraft: Xbox 360 Edition. Cabe destacar que tanto esta versión para la plataforma Xbox 360 como la de Android están limitadas en cuanto a funcionalidad en comparación con su versión para PC, entre otras características remarcables se puede decir que los mundos están limitados en cuanto a tamaño y el número de jugadores en línea máximos para Xbox 360 son 8. Como es de esperar también se reciben actualizaciones gratuitas de estas versiones.

Posteriormente, se lanzó una versión para la plataforma de videojuegos de Sony PlayStation, tanto portátil como consola en Noviembre de 2014 desarrollado en colaboración de 4JStudios. Actualmente presenta las mismas características que las versiones para consola.

En septiembre de 2014 Microsoft adquirió Mojang por 2500 millones de dólares y por tanto Minecraft, este hecho hizo que, su creador y cofundador de la empresa Mojang, Markus “Notch” Persson se distanciara de la compañía.

En la actualidad Minecraft sigue recibiendo actualizaciones, mejorando el rendimiento y añadiendo nuevas características cómo: nuevos jefazos, bloques, encantamientos, biomas y modos de juego como por ejemplo su modo de partida en red o Minecraft Realms.

2. Descripción

Minecraft es un videojuego de tipo supervivencia, construcción y no lineal. Se caracteriza por hacer uso de cubos que representan distintos materiales con los que el jugador puede interactuar para crear construcciones, en una versión minimalista de la realidad en 3 dimensiones. En este mundo virtual el jugador tiene que sobrevivir buscando y recolectando recursos, por medio de la minería, la caza, el cultivo, la pesca, la ganadería y la elaboración de objetos (crafting), con tal de sobrevivir al hambre y monstruos que este mundo virtual contiene.

3. Aspectos relevantes

1. Modos de juego

Minecraft dispone de los siguientes modos de juego, individual, multijugador y Minecraft Realms. A continuación una breve explicación de ellos.

Individual

En el modo de juego individual el jugador puede elegir entre tres opciones: Survival, Hardcore y Creative.

Survival

Este modo de juego se caracteriza por, cómo su propio nombre denomina, la supervivencia del jugador frente a las adversidades que Minecraft ofrece. El jugador deberá buscar, conseguir y elaborar los recursos que son necesarios para su supervivencia, en caso de muerte del jugador conservaría sus construcciones pero no los ítems que llevara en el momento de su muerte, los cuales se caerían en el lugar de la muerte, además ofrece la opción de poder guardar el estado de la partida actual y poder retomarlo en cualquier momento.

Esta modalidad además añade 4 modos de dificultad: pacífico, fácil, normal y difícil. Cabe destacar que si selecciona la dificultad pacifica los monstruos agresivos dejarían de serlo y el jugador solo se verá en peligro por ahogamiento, quemaduras, venenos, derrumbamientos o caídas. Las dificultades fácil, normal y difícil atienden a una escala exponencial de daño por lo que a mayor grado de dificultad mayor será el daño recibió tanto de las entidades como del entorno.

Hardcore

Este modo de juego es un calco de la versión Survival con unas pequeñas diferencias que lo caracterizan y ofrecen una dificultad añadida al jugador. En este modo la vida es limitada, por lo que si el jugador muere perderá todo su progreso y deberá empezar el juego en un mundo totalmente nuevo sin conservar nada de la partida actual, además la dificultad que ofrecen los enemigos es mucho mayor que en el modo Survival.

Creativo

Este modo de juego se caracteriza por dar al jugador acceso ilimitado a todos los elementos del juego, permite al jugador volar y poder destruir bloques instantáneamente con el fin de poder construir estructuras o mecanismos de una manera fácil y sin las incomodidades que ofrecería en modo Survival.

Una funcionalidad de los modos de juego individuales anteriormente citados es que pueden ser compartidos con otros jugadores en red de área local, siendo el computador que aloja el juego un cliente/servidor al mismo tiempo y los que son compartidos clientes del servidor del área local.

Multijugador

La versión multijugador de Minecraft permite al jugador interactuar con otros jugadores que se conecten al mismo servidor desde cualquier parte del mundo y dependiendo del tipo de servidor disfrutar de las características originales del juego en sí (Servidor oficial o Vanilla) o ser modificadas por complementos o modificaciones que añaden nuevas funcionalidades a los servidores (Servidores no oficiales o mods).

Los servidores de Minecraft, oficiales y no oficiales, son sensibles a estas modificaciones, lo que contribuye a que cada servidor, desde el punto de vista del administrador, pueda ser modificado al gusto y con la limitación de las modificaciones y complementos existentes, y desde el punto de vista del jugador un entorno y dinámica totalmente distinta. (Véase Anexo II para una descripción más detallada de los servidores y complementos)

4. Mobs

A continuación una lista de las posibles entidades o monstruos, comúnmente llamados mobs, que se pueden encontrar en Minecraft. Se diferenciaran por su nivel de hostilidad e irán acompañados de una breve explicación de su comportamiento:

- Entidades Pasivas
    - Aldeano (villager en inglés): Es una entidad pasiva, se puede comerciar con él mediante el intercambio de objetos y dependiendo del trabajo que desempeñe comerciará un tipo de objetos. El Aldeano puede tener distintos trabajos: granjero, bibliotecario, carnicero, herrero, monje; y se diferencian según el vestido que lleven.
    - Gallina (*chicken* en inglés): Es una animal pasivo, que en ocasiones deja huevos en el suelo. No sufre daño por caídas puesto que puede planear. El jugador puede obtener plumas y la carne del mismo si lo mata.
    - Cerdo (*pig* en inglés): Es un animal pasivo del cual se puede obtener carne si se mata.
    - Oveja (*sheep* en inglés): Es un animal pasivo, el jugador puede tintarlas y esquilarlas con tal de conseguir bloques de lana de colores. El jugador puede obtener carne si la mata y si no ha sido esquilada además la lana.
    - Vaca (*cow* en inglés): Es un animal pasivo. El jugador puede obtener leche de ellas si las ordeña, carne y cuero si las mata.
    - Vaca-seta (*mushrom* cow en inglés): Es un animal pasivo que se puede encontrar en el bioma “isla de setas”, al ordeñarla produce sopa de setas y al matarla dejan caer setas, carne y cuero.
    - Ocelote (*ocelot* en inglés): Es un animal pasivo que se encuentra en el Bioma de Jungla, pueden ser atraídos mediante pescado crudo y no dejan caer nada al morir. Se pueden domesticar y cambian su apariencia a un gato común al hacerlo.
    - Conejo (*rabbit* en inglés): Es un animal pasivo caracterizado por dar saltitos para moverse, es un animal escurridizo y difícil de matar. Si se mata se puede obtener cuero, carne y patas de conejo, estas últimas son usadas en pociones. Como excepción existe una versión agresiva de este animal, llamado *KillerBunny* o conejo asesino que no dará tregua al jugador.
    - Murciélago (bat en inglés): Es un animal pasivo que se encuentra en cuevas y se puede encontrar, o bien, en reposo o volando. No deja caer nada al morir.
    - Calamar (*squid* en inglés): Es un animal pasivo que se puede localizar en casi cualquier masa de agua natural. Si se mata puede dejar caer tinta negra.
    - Caballo (*horse* en inglés): Es un animal pasivo que se puede domesticar y convertirse en la montura del jugador, mediante una silla de montar. Además pueden tener armadura que reducirá el daño recibido por otras entidades agresivas que ataquen al jugador. Como curiosidad existen dos caballos ocultos solo generables mediante comando con la apariencia que su propio nombre indica: caballo esqueleto y caballo zombi.
    - Burro (*donkey* en inglés): Es un animal pasivo similar al caballo. No pueden llevar armadura.

    - Mula (*mule* en inglés): Es un animal pasivo similar al caballo que además puede transportar alforjas y en ellas los objetos del jugador. No pueden llevar armadura.
- Entidades neutrales
    - Golem de nieve (*snow* golem en inglés): Es una entidad neutral que solo se puede generar mediante comando o por acción del usuario. Lanzan, con tal de mantener alejados a otros monstruos, bolas de nieve que no hacen daño ninguno.
    - Golem de hierro (*iron* golem en inglés): Es una entidad neutral que aparece de manera natural en las aldeas de aldeanos para su protección frente a las amenazas que puedan ser otros monstruos hostiles. Su manera de proteger consiste en lanzar a sus objetivos por el aire. El jugador puede crear sus propios golems mediante bloques de hierro y una calabaza.
    - Araña y araña de cueva (spider y cave spider en inglés, respectivamente): Son monstruos neutrales que solo atacaran agresivamente si es de noche o durante el día si son atacadas. Al morir dejan caer los ojos y telas de araña. Las arañas de cueva son, además, venenosas y más pequeñas que las arañas normales.
    - Lobo (*wolf* en inglés): Es una entidad neutral que como el Ocelote también es domesticable. Normalmente, estas criaturas se encuentran en manadas en los bosques y si se ataca a una la manada la defenderá de su agresor atacándola.
    - Oso polar (polar *bear* en inglés): Es una entidad neutral que vaga por los parajes nevados. Se pueden sentir amenazadas si tienen una cría cerca y atacar sobre sus patas traseras. Al igual que el caballo puede ser usado como montura. Es una de las últimas criaturas introducidas en Minecraft (versión 1.10).
    - *Enderman* (mismo nombre en inglés): Es una entidad neutral que no atacará a menos que se haga contacto visual, es capaz de tele-portarse y es dañado por el agua (lluvia o emplazamientos de agua), además es capaz de coger bloques y cambiarlos de sitio. Al morir deja caer “ojos de ender” (Ender *eyes*), capaces de tele-transportar al jugador al lugar que son lanzados.
    - Hombre cerdo zombi (*zombie pigman* en inglés): Es una criatura neutral que como su nombre indica es una mezcla entre un hombre, un cerdo y un zombi. Al igual que los lobos si es atacado uno, todas las criaturas de su mismo tipo en un radio cercano atacaran al agresor. Suelen aparecer de manera natural en el mundo Nether, aunque, también fortuitamente, si un rayo cae sobre un cerdo en el mundo normal se convertirá en un hombre cerdo zombi. Al morir suele soltar el arma que lleva equipada y pepitas de oro.
- Entidades hostiles
    - Zombi (zombie en inglés): Es una criatura hostil que normalmente aparece de noche, puesto que de día y al contacto con el sol empezará a arder. Al morir suele soltar carne podrida. Como curiosidad existe una versión de zombi aldeano que puede ser, mediante pociones, convertido de nuevo en un Aldeano. En la versión 1.10 de Minecraft han introducido

un nuevo tipo denominado zombi pusilánime que aparecen de manera natural en los desiertos y no son afectados por la luz solar.

- Creeper (mismo nombre en inglés): Es una entidad hostil que se puede encontrar en cualquier lugar, utilizan el sigilo para posicionarse silenciosamente detrás del jugador y explotan pasados unos segundos, con un siseo que indica la cuenta atrás. Al explotar mueren y dejan un hueco haya donde se produjo la explosión. Al morir sin llegar a explotar suelen soltar pólvora y si son fortuitamente impactados por un rayo la explosión se hace aún mayor y el daño aumenta.
- Esqueleto (*skeleton* en inglés): Es una entidad agresiva que lanza flechas con un arco. Al igual que los zombis normales aparecen con la noche y de día buscan refugio a la sombra o son quemados por el Sol. Al morir suelen soltar huesos, flechas y el arco que tenía equipado. En la versión 1.10 de Minecraft se introdujo una variante de esqueletos, que aparecen en los biomas helados, denominado “Errante” con aspecto fantasmagórico.
- Lepisma (*silverfish* en inglés): es una entidad hostil que se esconde en bloques de las cuevas, normalmente en las fortalezas. Al picar estos bloques aparecen varios y atacan al jugador, para su ventaja además son pequeños y difíciles de acertar.
- *Endermite* (mismo nombre en inglés): Similar al lepisma esta criatura es hostil, es posible que aparezcan cuando un *enderman* se tele transporta o cuando un jugador hace lo mismo con un ojo de ender.
- *Slime* y Cubo de Magma (Slime y *Magmacube* en inglés): son entidades hostiles de forma cúbica que aparecen en zonas pantanosas y en el mundo Nether, respectivamente. Se caracterizan por caminar a saltos e intentar aplastar a su oponente. Al ser atacados se separan en criaturas más pequeñas y el cubo de magma puede además quemar al oponente. Al morir suelen soltar bolas de *slime* y bolas de crema de magma respectivamente.
- Bruja (*witch* en inglés): es una criatura hostil de aspecto similar a un aldeano con la diferencia que lleva un gorro en cono que la delata. Al atacar lanza pociones de ralentización, veneno y daño instantáneo. Al morir puede soltar pociones, botellas vacías y pólvora entre otros objetos.
- *Blaze* (mismo nombre en inglés): Es una criatura hostil del mundo Nether, se caracteriza por atacar lanzando llamaradas de fuego y flotar en el aire.
- *Ghast* (mismo nombre en inglés): Es una criatura hostil del mundo Nether, vuela y lanza bolas de fuego desde una distancia alejada. Tienen un sonido característico similar a un llanto. Al morir suele soltar “lagrima de *ghast*” un objeto necesario para realizar alguna poción.
- Esqueleto wither (wither skeleton en inglés): es una criatura hostil similar al esqueleto normal pero aparece en el mundo Nether, con la diferencia de que porta equipada una espada y al atacar el oponente roba su vida. Al morir hay una pequeña probabilidad de que suele su cabeza, con la cual se puede invocar un Jefe llamado “wither”.
- *Shulker* (mismo nombre en inglés): es una criatura hostil que aparece en las fortalezas y barcos del mundo “TheEnd”. En apariencia es un bloque morado que cuando detecta a un oponente se abre y lanza proyectiles

que provocan daño y levitación. Solo es posible atacarlo cuando está abierto.
    - Guardián (mismo nombre en inglés): es una criatura marina hostil que se encuentra en fortalezas submarinas, ataca mediante un rayo que debilita a su oponente.
- Jefazos
    - Dragón del fin (*EnderDragon* en inglés): Es el primer jefazo introducido en el desarrollo de Minecraft. Es una criatura hostil que ronda el mundo TheEnd y ataca mediante picados al jugador que se presente. Su dificultad radica en unos faros que revitalizan al dragón cuando este está próximo a ellos. Al morir suelta mucha experiencia, crea un portal de vuelta al mundo normal y deja un huevo de dragón meramente decorativo.
    - Wither (mismo nombre en inglés): Es el segundo jefazo que fue introducido en Minecraft. Es una criatura hostil con tres cabezas que lanza cabezas explosivas como proyectiles, vuela hasta que su vida es mínima, por lo que es necesario derribarlo primero. Como el esqueleto wither este roba la vida a sus víctimas. Además rompe casi todos los bloques si entra en contacto con ellos.
    - Guardián anciano (Elder *guardian* en inglés): Es el tercer jefazo introducido en Minecraft. Es una criatura hostil con el mismo comportamiento que el guardián normal, pero de mayor tamaño y mayor daño. Al morir deja caer peces y un bloque de esponja.

5. Mundos

A continuación explicaré unos conceptos previos antes de explicar los tipos de mundos que existen en Minecraft, entre ellos: el generador de mundos, en concepto chunk y biomas.

Generador de mundos

Cuando una persona escucha mundo, espera ver un mundo finito, en Minecraft no es posible. Fue en la versión anterior a la alfa (versión Infdev nombre de versión reducida de Infinite Development) cuando Notch decidió reescribir el código del videojuego para permitir la generación de terrenos infinitos. Más tarde, en la versión Beta 1.3 de Minecraft se introdujo la opción de poder elegir una semilla que generaría los mundos. Las semillas son números aleatorios dentro de un rango que generan un mundo de una forma, de modo que si se usa esa misma semilla en cualquier otro lugar se obtendría el mismo resultado. En la actualidad pocos cambios ha recibido el generador de mundos de Minecraft, solo los necesarios para evitar eventos ilógicos en la generación de mundos, como pueden ser islas flotantes en el aire.

Chunk

Un Chunk, como la propia palabra significa, es una porción de un mapa, cada chunk está limitado por el limite más alto y el limite más bajo del propio mundo (0 y 256, respectivamente) y tiene 16 bloques de ancho y de largo, se crean dependiendo de la distancia de visión del jugador y de si juega en el modo un jugador o multijugador, donde esta magnitud puede variar según la configuración del servidor. Se crean mediante el anterior citado generador de mundos, solo en el caso de que no existieran antes, si estos ya existían son cargados en el juego. Del mismo modo que se cargan unos chunks por acción del jugador, los que dejan de estar en su distancia de visión son descargados

para no copar la memoria.

Biomas

En Minecraft también existen los biomas, al igual que en el mundo real pero en una versión mucho más reducida, entre ellos se puede encontrar: bosque, desierto, llanura, pantano, jungla, tundra, taiga, montañas, océano; e islas de setas, esta última es propia del juego.

Tipos de mundo

En el modo de juego un solo jugador se puede crear distintos tipos de mundos según nos interese. A continuación una explicación de cada uno de ellos:

- Predeterminado: Este tipo genera un mundo con una semilla obtenida del reloj del ordenador.
- Extraplano: Este tipo de mundo, se crea totalmente llano con una capa superior de hierba, 2 capas de tierra y una última de *bedrock*, que evita la caída del jugador al exterior del mundo.
- Biomas grandes: Este tipo al igual que el predeterminado genera una semilla aleatoria y además agranda por un factor 4 en ancho y largo los biomas del mundo predeterminado.
- Amplificado: Este tipo en contrapartida con el de biomas grandes genera una semilla aleatoria y además agranda la altura y profundidad de los accidentes geográficos.
- Personalizado: Este tipo permite la introducción de la semilla manualmente y además cuenta con 4 páginas de configuración que nos permitirá realizar una selección más detallada de los aspectos del mundo que se desee.
- Debug mode: Este tipo de mundo, no se puede jugar, simplemente es una representación de todos los bloques de Minecraft en todas sus posibles orientaciones, el jugador no podrá ni romper ni colocar bloques, solo podrá volar para inspeccionarlos.

Además en el modo de juego supervivencia existen 2 mundos independientes de los generales, comentados anteriormente, que son accesibles mediante portales desde el mundo en el que se empieza a jugar. Estos son: Nether y The End.

- Nether (Inferior en español): simula ser un infierno de otra dimensión, se accede a través de un portal que el jugador debe construir mediante obsidiana y prenderle fuego para activarlo. Es un mundo lúgubre, con lagos inmensos de lava y con fortalezas plagadas de criaturas poderosas y de aspecto demoniaco. Está acotado por bedrock en la altura 127 y en lo más profundo (altura 0) por bedrock. Se genera con la misma semilla con la que el mundo normal es generado.
- The End (El fin en español): es otra dimensión, que al igual que el Nether se accede mediante un portal, este portal se encuentra en una fortaleza en el mundo normal y es activado mediante “ojos de ender”. Se caracteriza por ser una gran isla central rodeada de otras islas más pequeñas que flotan en el vacío, están habitadas por endermans y el jefazo *Ender Dragon*. En la versión 1.9 fue introducido en Minecraft las ciudades, barcos y la criatura Shulker en mundo The End, ampliando el mundo que en principio era una simple isla central con el jefazo *Ender Dragon*.

6. Aspecto visual

Un aspecto importante, es el aspecto visual del videojuego, de carácter retro y de texturas pixeladas Minecraft es con diferencia un juego de estilo indie que llama la atención por su sencillez y su capacidad para ser modificado visualmente. Con la introducción de una modificación del juego original por parte del equipo de Mojang que permite el cambio de paquetes de texturas y sonidos en el videojuego original.

2 Anexo II Servidores y complementos

1. Servidores

1. Historia

En Junio de 2009 se lanza la versión Minecraft con el modo multijugador, que básicamente era una versión multijugador de supervivencia o las siglas SMP (Survival multiplayer en inglés). Notch planeó desarrollar un API mod para Abril de 2011, con la que se podrían realizar modificaciones que añadieran funcionalidades y nuevos contenidos al juego. En paralelo, los jugadores empiezan a formar una comunidad de desarrollo de mods para Minecraft entre ellos: hMod Project, Bukkit Project, Minecraft Forge y Minecraft Coder Pack. Jeb Bergensten, apodado Jeb, se convirtió en jefe de desarrollo de Minecraft en Diciembre de 2012 y declaró que el desarrollo de un API mod para Minecraft sería su máxima prioridad. A pesar de las intenciones tanto de Jeb y Notch a día de hoy no se han cumplido. Aun así, se tuvo gran relación entre los equipos de Mojang y Bukkit para el desarrollo de una API oficial, pero no tardo en desbaratarse este objetivo de futuro cuando un desarrollador del equipo Bukkit desmanteló todo el proyecto Bukkit. Este desarrollador denunció de mala fe la vulneración de sus derechos como autor de ciertas partes del código del servidor modificado Craftbukkit, tipificada por la ley Digital Millenium Copyright Act (DMCA) de Estados Unidos. A día de hoy el trabajo de Bukkit es inaccesible, aunque se conservan los foros y los complementos creados por la comunidad.

Tras estos hechos Bukkit fue sucedido por Spigot en Junio de 2012, la cual es una versión bifurcada de Bukkit, es compatible con la mayoría de los complementos de Bukkit y tiene una comunidad de más de 65000 miembros. También sufrió en Septiembre de 2014 un desmantelado de las descargas, pero encontraron la solución para poder distribuir su mod mediante el uso de parches binarios. Comprometidos con la causa y alegando que no renunciarían a todo el trabajo realizado a causa de un único desarrollador, continúan con su tarea de mejorar en base a las actualizaciones que recibe el juego original.

2. Oficial vs no oficial

Los servidores oficiales, comúnmente denominados Vanilla, ofrecen la posibilidad de jugar el modo de supervivencia con muchos jugadores, en cambio servidores no oficiales añaden, además de la funcionalidad de los servidores oficiales, la posibilidad de optimizar el juego mediante la configuración de detalles, la adición de complementos, mejoras de rendimiento. En este caso se citarán algunos de los que Spigot aporta:

- Incremento de los ticks por segundo.
- Optimización del crecimiento, descarga y carga de Chunks.
- Mensajes configurables propios de las versiones Vanilla.
- Compatibilidad con complementos de Bukkit
- Garbage collector de Chunks para prevenir las pérdidas de Chunks.
- Activación de entidades y seguimiento de rangos para asegurar que los recursos tanto de la parte servidor como la del cliente son usadas solo cuando es necesario.
- Control preciso de crecimiento de cultivos, Chunks y ticks de reloj.

3. Instalación de un servidor modificado

En este apartado se hará una breve explicación de los elementos necesarios para la instalación de un servidor no oficial Minecraft, los pasos necesarios de instalación y configuración de un servidor y como conectar con el mismo.

Computador o servidor

A día de hoy los servidores de hospedaje de juegos traen consigo una interfaz de configuración para poder realizar la instalación de nuestro servidor sin tener que depender de los siguientes elementos que se citarán, puesto que ya están en el propio servidor, aunque es posible modificar la versión del servidor de Minecraft que parezca conveniente. Una instalación local de un servidor Minecraft obliga a tener los restantes elementos que se citarán a continuación.

Lanzador del servidor

Es un archivo que se encarga de lanzar el servidor Minecraft en el computador o servidor en el que este alojado. Este archivo es dependiente del sistema operativo en el que se encuentre, por lo que se configurará de una manera u otra (Windows, Mac OS X, Linux).

Servidor.

El archivo es de por si el servidor, tiene extensión jar y una versión concreta. La elección de la versión que se elija para el servidor tiene una repercusión directa en la versión de los clientes de Minecraft que se conectarán y de los complementos que se añadirán con tal de extender la funcionalidad, por lo que se deberá elegir con previsión que complementos se usarán. El servidor, como se avanzó, deberá ser descargado solo en el caso de que el servidor de hospedaje de juegos no lo contenga o si se trata de una instalación local.

Java

Minecraft requiere Java 7 o superior para su ejecución.

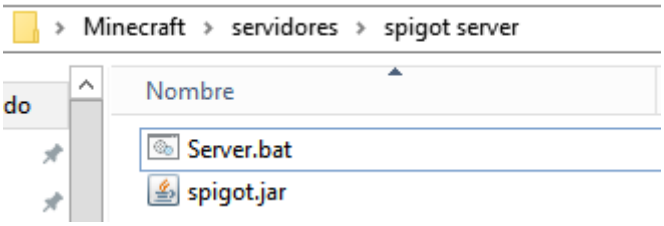
Pasos

La instalación del servidor no oficial de Minecraft es dependiente del sistema operativo. Se obviará en el caso en el que se usa un servidor especializado en el hospedaje de juegos, dado que suelen tener las herramientas necesarias para su instalación y configuración, en el caso que no lo tenga se siguen los mismos pasos que en una instalación en un computador local.

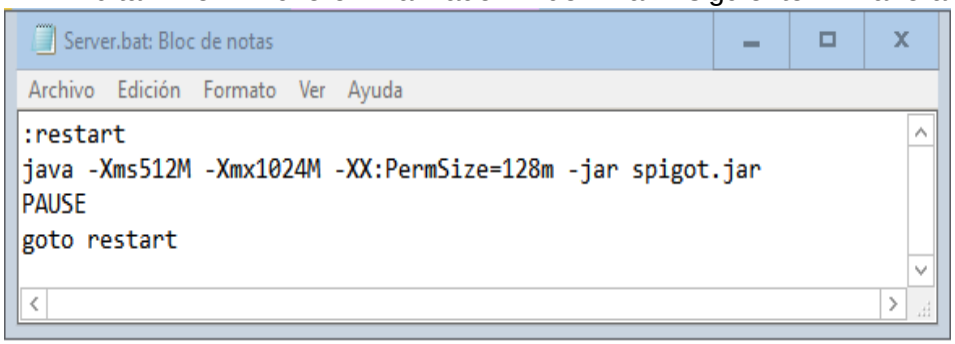
1º Descargar el servidor no oficial de Minecraft que se desee instalar. En el ejemplo se usará Spigot 1.8.7 en el sistema operativo Windows en el que se podrán ver con mayor detalle los pasos acompañados de imágenes.

2º Dependiendo del sistema operativo:

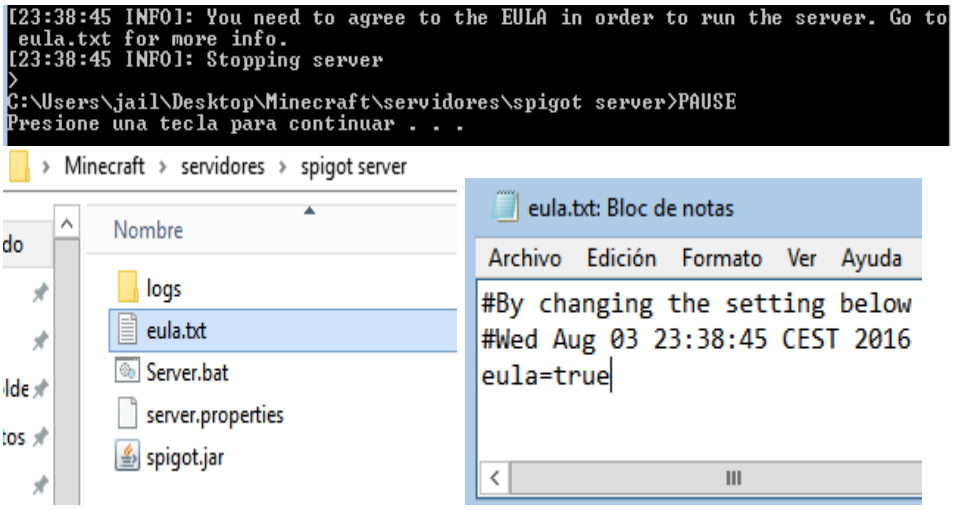
- Windows
  - 1º Crear un fichero lanzador de extensión bat en la misma carpeta en la que se encuentre el servidor. Se recomienda que el servidor y el lanzador



    se encuentren en una misma carpeta, puesto que se crearán automáticamente ficheros en ella. En el ejemplo se ha creado la carpeta "Spigot server".
  - 2º Editar el fichero lanzador de la siguiente manera

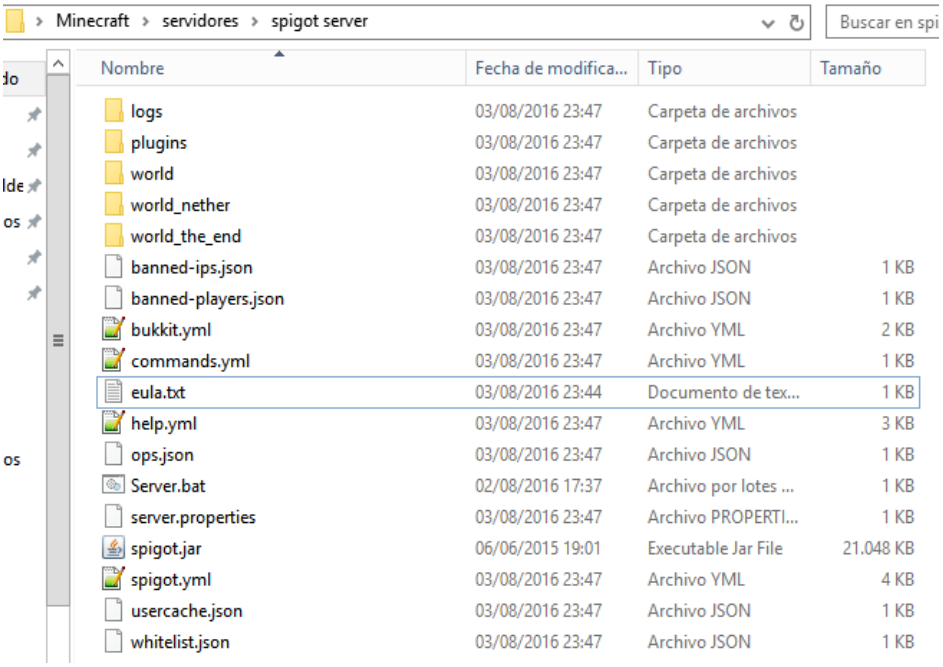


    Las opciones –Xms y –Xmx indican, respectivamente, el mínimo y máximo consumo de memoria RAM permitido, se puede aumentar o disminuir si es necesario. La opción –XX: PermSize indica el espacio reservado para las definiciones de clases y métodos compilados. Si se trabaja sobre Java 8 no es necesaria.
  - 3º Se ejecutará por primera vez el archivo de extensión bat y esperar a que se pare el servidor. Se deberá aceptar la EULA para completar la instalación del servidor, para ello, en la carpeta del servidor se abrirá el archivo eula.txt y se modificará la línea que pone "eula=false" por "eula=true".



- 3º Desde consola pulsar cualquier tecla para continuar con la instalación y esperar el siguiente mensaje en la consola de comandos:

El mensaje indica que el servidor se esta ejecutando y que se habrán creado los ficheros necesarios para su configuracion. Desde la consola es posible ejecutar comandos para parar, manejar el servidor y administrar complementos que hagan uso de comandos de consola. Así debería quedar la carpeta del servidor.



- 4º El siguiente paso es configurar el servidor, con las necesidades que sean necesarias, para ello se debe parar el servidor con la palabra “stop” en la consola y acceder a la carpeta donde esta el ejecutable y editar los siguientes ficheros:
  - server.properties
  - bukkit.yml
  - spigot.yml
- 5º Volver a iniciar el servidor y comprobar los cambios.

- Linux
  - 1º Crear script de extension sh en la misma carpeta en la que se encuentre el servidor.
  - 2º Editar el script lanzador del servidor de la siguiente manera:

```
#!/bin/sh

java -Xms512M -Xmx1024M -XX:MaxPermSize=128M -jar spigot.jar
```

  - 3º Abre un terminal y ejecuta lo siguiente en el mismo directorio:

```
chmod +x start.sh
```

  - 4º Lanzar el script para iniciar el servidor. Al igual que en la instalación del servidor en Windows el terminal será la consola del servidor.
  - 5º Parar el servidor, configurar e iniciar para comprobar los cambios. Los ficheros de configuración son los mismos que en Windows.
- Mac OS X
  - 1º Crear script de extension command en la carpeta en la que se encuentre el servidor.
  - 2º Editar el script lanzador del servidor de la siguiente manera:

```
#!/bin/sh

cd "$( dirname "$0" )"
java -Xms512M -Xmx1024M -XX:MaxPermSize=128M -jar spigot.jar
```

  - 3º Abre un terminal y escribe lo siguiente en el mismo directorio sin pulsar la tecla enter:

```
chmod a+x
```

  - 4º Arrastra el script hasta el terminal abierto.
  - 5º Inicia el servidor haciendo doble click sobre el script. Al igual que en los demás sistemas operativos el terminal será la consola del servidor.
  - 6º Parar el servidor, configurar e iniciar comprobando los cambios. Los ficheros de configuración son los mismos que en Windows.

2. Complementos

1. Descripción

Un complemento o plugin en inglés, referido a servidores Minecraft, es un elemento software diseñado para añadir, modificar o eliminar comportamientos o funcionalidades que antes no existían en Minecraft. Existe una gran amalgama de tipos, pueden dedicarse a la gestión de usuarios, mini juegos, aspectos de rol (misiones, comercio, sistema monetario, rangos, clanes o facciones), adición de funcionalidades (ayudas al jugador, menús navegables) o sistemas anti trucos, entre otros.

Estos complementos se pueden encontrar en las comunidades de Bukkit y Spigot, entre otras y son desarrollados por miembros de las propias comunidades. Cualquiera que tenga conocimientos de Java es capaz de realizar un complemento para Minecraft.

2. Aspectos importantes de los complementos

Compatibilidad

Los complementos desarrollados con Bukkit pueden ser compatibles con Spigot, pero no a la inversa. Además son dependientes de la versión en la que el servidor implementa, por lo que, para su desarrollo se debe usar siempre la misma versión que usa el servidor.

Archivos de configuración

Los complementos tras su instalación en el servidor crean una carpeta con el nombre del plugin, en la que se guardarán los archivos de configuración y de datos. El método de configuración de los complementos en servidores de Minecraft está basado en ficheros de configuración Yaml, estos ficheros son archivos de texto plano que determinan la configuración de un complemento, en ellos se definen pares propiedad-valor que usa el propio complemento para modificar su comportamiento y suelen estar acompañados, en ocasiones, por comentarios explicativos sobre la configuración. En otros casos es necesario realizar una visita a la web donde se publican estos complementos para obtener ayuda adicional, lo que puede resultar un poco tedioso para administradores poco expertos en servidores de esta índole.

Existen complementos que pueden ser configurados dentro del juego mediante comandos, estos modifican la configuración y pueden ser reiniciados en caliente en el servidor mediante un comando propio del complemento. Sino es el caso se deberá acceder a los archivos de configuración, mientras el servidor está inactivo y modificar lo necesario para volver a iniciarlo.

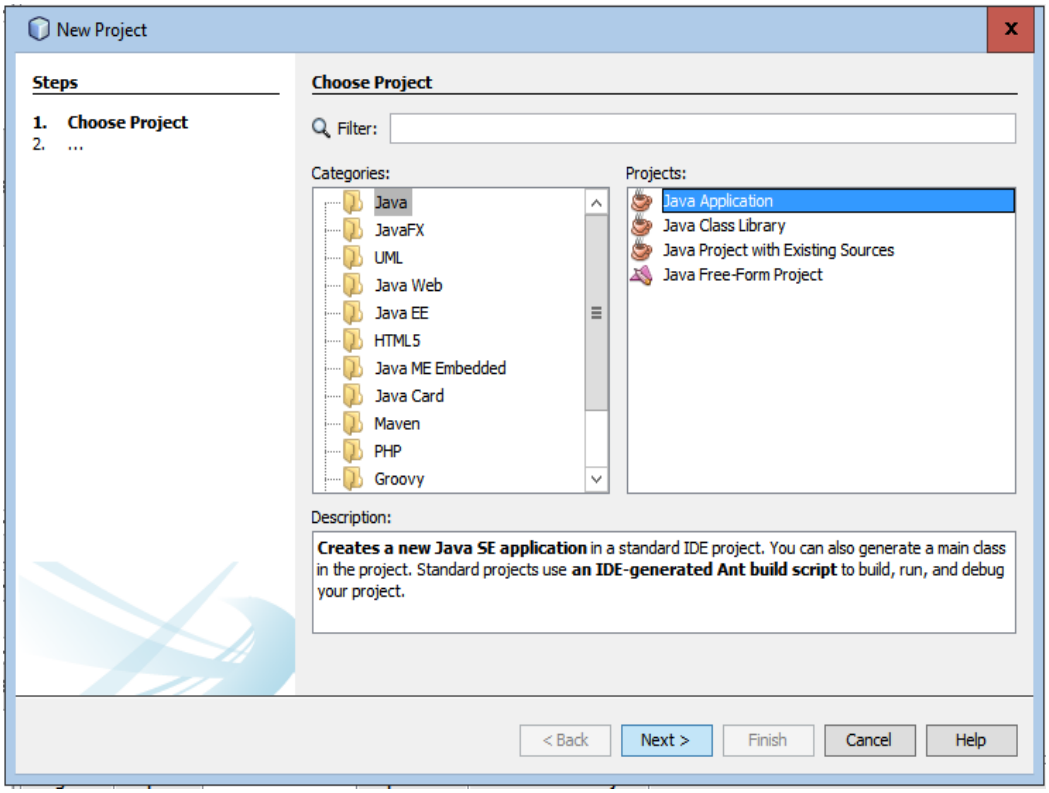
3. Requisitos para el desarrollo de complementos

- Java: El desarrollo de complementos se realiza mediante el lenguaje de programación orientado a objetos Java y es la base indispensable para empezar a desarrollarlos.
- Entorno de desarrollo: Los entornos de desarrollo son fundamentales para el desarrollo de complementos. Integran herramientas que son de especial utilidad, como: bases de datos, herramientas de autoformato o autocompletado de código, repositorios, testing, entre otras.
- API: La librería esencial en todo complemento desarrollado para Minecraft.
- Javadoc de la API: Herramienta imprescindible que merece tener cerca cuando se desarrollan complementos.
- Conocimientos básicos: Estructura básica de un complemento Minecraft, captura de eventos (listeners) y comandos, ficheros de configuración, permisos, tareas, manipulación de bloques, inventarios, ítems, metadatos, bases de datos, entre otros.

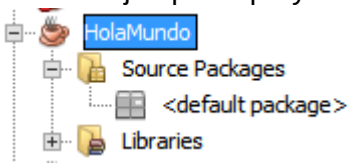
4. Desarrollo de un complemento “Hola mundo”

Con los requisitos, arriba citados, se desarrollará un complemento ejemplo para un servidor Spigot versión 1.8.7, se hará uso de la API Spigot de la misma versión y el entorno de desarrollo integrado Netbeans 8.0.2.

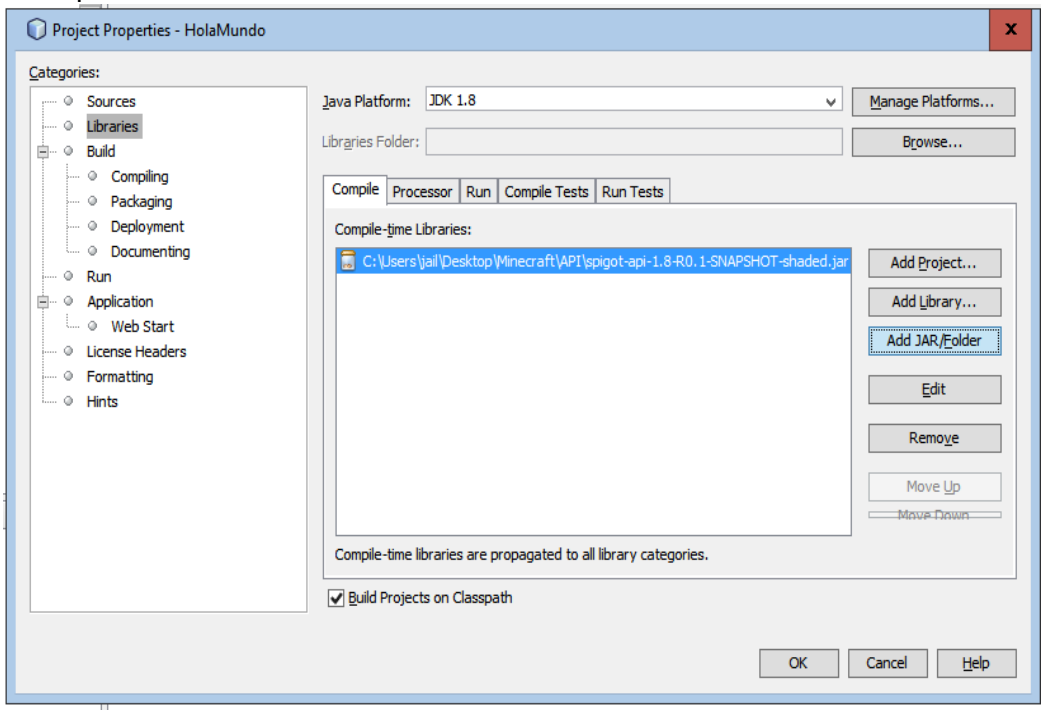
- 1º Crear proyecto nuevo Java en Netbeans



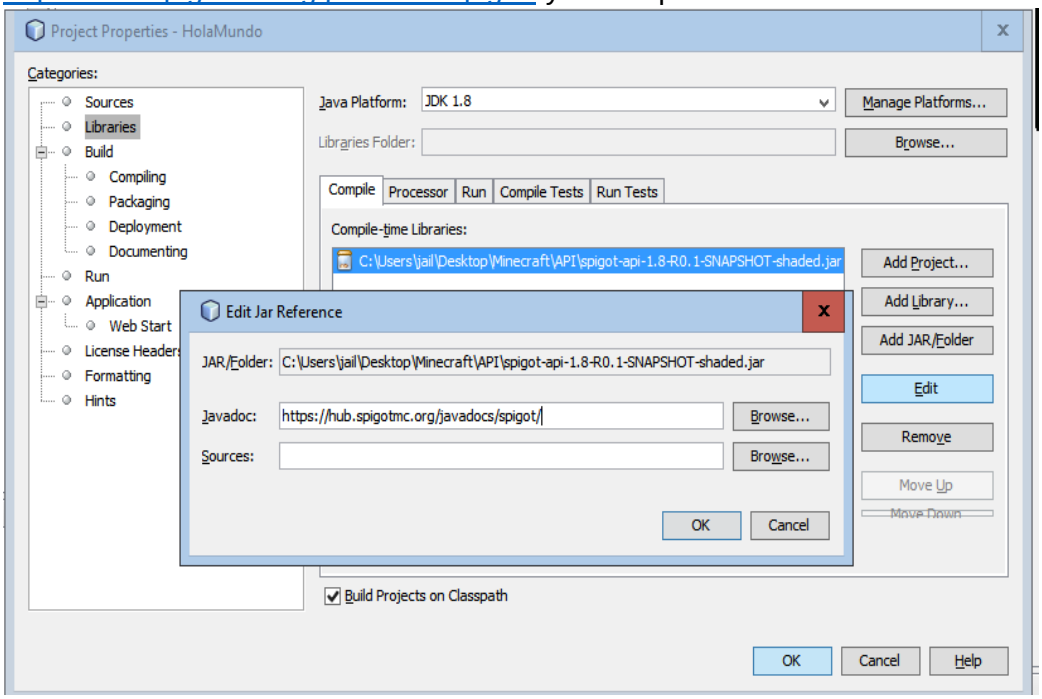
En este ejemplo el proyecto se llamará “HolaMundo”.



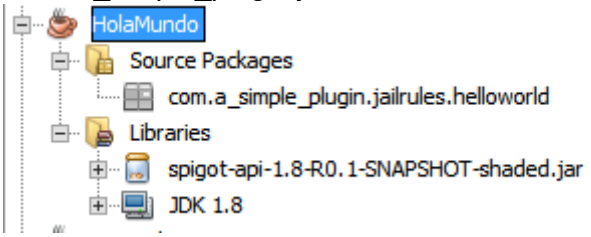
- 2º Acceder a las propiedades del proyecto y añadir la API de Spigot en la pestaña “Compile”.



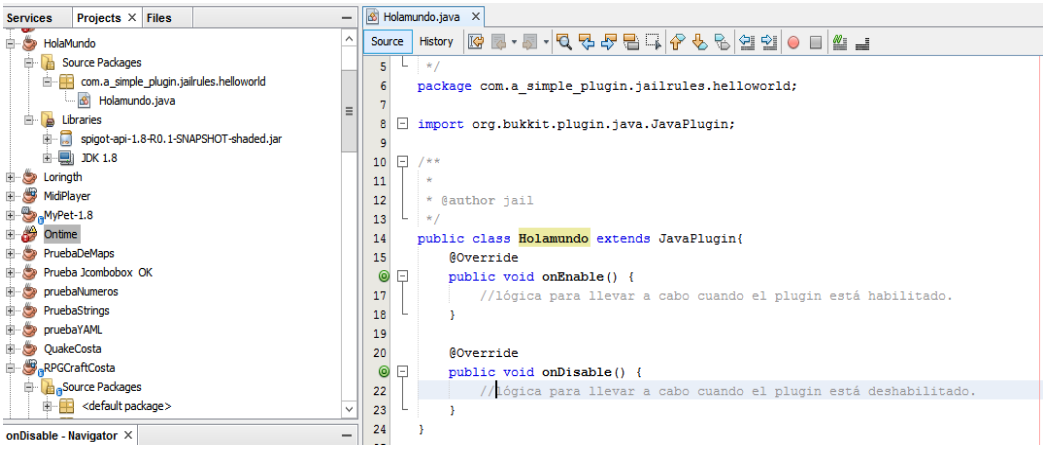
Se podrá asociar además el Javadoc a la API por comodidad. Seleccionada la API dar click en el botón Edit y rellenar la caja de texto Javadoc con https://hub.spigotmc.org/javadocs/spigot/ y se acepta hasta volver a la interfaz.



- 3º Crear un paquete en el proyecto. Por convención, si es propietario de un dominio, se debe crear con el nombre del dominio a la inversa, por ejemplo si el dominio es a_simple_plugin.com el nombre del paquete sería “com.a_simple_plugin”. Si el dominio lo administra mas de una persona se agregará el apodo y por ultimo el nombre del proyecto.En este caso como es un ejemplo pequeño se usará el ejemplo anterior acompañado de un apodo y el nombre del proyecto. Sería de este modo “com.a_simple_plugin.jailrules.helloworld”.



- 4º Crear la clase principal del complemento que extenderá a la clase JavaPlugin, se llamará “Holamundo”. Se escribiran los métodos que deben existir obligatoriamente para insertar la lógica que tendrá al iniciar y apagar el complemento, estos métodos son onEnable() y onDisable().



- onEnable() se ejecuta al cargar las librerias del servidor, se deberá inicializar los elementos que el complemento usará durante la ejecución.
- onDisable() se ejecuta al descargar las librerias, mediante un apagado o reinicio del servidor Minecraft, si con el complemento se realizan modificaciones o se guarda información de los jugadores en este metodo se deberá guardar cualquier cambio producido para el proximo inicio del servidor.

- 5º Introducir mensajes de consola en los metodos onEnable() y onDisable() para cerciorarse que el complemento se activa o se desactiva cuando el servidor se enciende y se apaga, respectivamente.

```
24      @Override
25      public void onEnable() {
26          //lógica para llevar a cabo cuando el plugin está habilitado.
27          getLogger().info("Iniciando ejemplo HolaMundo");
28
29          getLogger().info("Ejemplo HolaMundo iniciado");
30
31      }
32
33      @Override
34      public void onDisable() {
35          //lógica para llevar a cabo cuando el plugin está deshabilitado.
36          getLogger().info("Desactivado ejemplo HolaMundo");
37      }
```

- 6º Implementar la funcionalidad del complemento. Se trata de un pequeño ejemplo por lo que se hará uso de un *listener* y un comando.
  - Para poder usar un listener se deberá implementar la clase Listener en la clase en la que se desee capturar los eventos. En este ejemplo la clase principal hará de captador de eventos.

```
13  * @author jail
14  */
15  public class Holamundo extends JavaPlugin implements Listener{
16      @Override
17      public void onEnable() {
```

Se capturará el evento de BlockBreakEvent que escucha cuando un bloque en el juego ha sido roto, de modo que avisará al jugador mediante un mensaje que la acción que realiza no está permitida. Para que el

listener haga efecto el método debe estar precedido de @EventHandler.

```
28
29          @EventHandler
30          public void onBlockBreakEvent(BlockBreakEvent event){
31              //Insertar logica
32          }
```

A continuación se implementará la acción que consistirá en dos partes, por un lado se avisará al jugador de que esa acción no esta permitida y por otro lado se cancelará la acción.

```
30      @EventHandler
31      public void onBlockBreakEvent(BlockBreakEvent event){
32          //ENVIAR MENSAJE AL JUGADOR
33          event.getPlayer().sendMessage(ChatColor.RED+"No esta permitida la acción que estas realizando");
34          //CANCELAR EL EVENTO
35          event.setCancelled(true);
36      }
```

Para finalizar se debe registrar el listener en la clase principal del plugin cuando el complemento es iniciado, en el metodo onEnable() en este caso:

```
24      @Override
25      public void onEnable() {
26          //lógica para llevar a cabo cuando el plugin está habilitado.
27          getLogger().info("Iniciando ejemplo HolaMundo");
28          //REGISTRAR EL LISTENER DE LA CLASE PRINCIPAL
29          this.getServer().getPluginManager().registerEvents(this, this);
30          getLogger().info("Ejemplo HolaMundo iniciado");
31      }
```

Si los listeners implementados se encuentran en otras clases se deberan inicializar con su constructor, por ejemplo si nuestro listener estuviera en una clase que se llama Oyentes al registrar el listener sería de la siguiente manera:

```
@Override
public void onEnable() {
    //lógica para llevar a cabo cuando el plugin está habilitado.
    getLogger().info("Iniciando ejemplo HolaMundo");
    //REGISTRAR EL LISTENER DE LA CLASE PRINCIPAL
    this.getServer().getPluginManager().registerEvents(new Oyentes(), this);
    getLogger().info("Ejemplo HolaMundo iniciado");
}
```

- Para poder crear un comando se necesitará implementar la clase CommandExecutor en la clase que se desee crear los comandos. En este caso se crearán los comandos en la clase principal del comando:

```
20  public class Holamundo extends JavaPlugin implements Listener,CommandExecutor{
```

Y se añadirá el metodo prototipo de captura de comandos precedido de la anotación @Override:

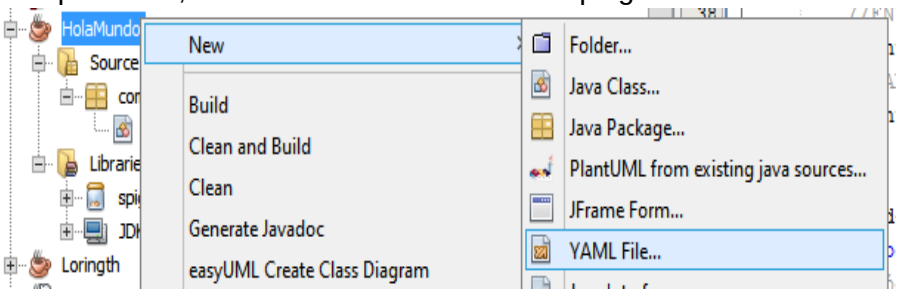
```
44      @Override
45      public boolean onCommand(CommandSender sender, Command cmd, String label, String[] args) {
46          //lógica de comandos
47          return true;
48      }
```

Los comandos introducidos por el chat van precedidos de el carácter barra o Slash en inglés (/). En este ejemplo se creará un comando u orden de nombre hola, para que cuando el jugador escriba /hola en el chat reciba una contestación por parte del servidor.

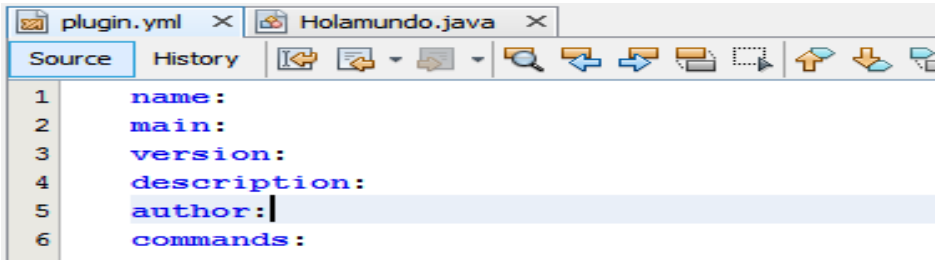
```java
@Override
public boolean onCommand(CommandSender sender, Command cmd, String label, String[] args) {
    //lógica de comandos
    if(sender instanceof Player){
        if(label.equalsIgnoreCase("hola")){
            if(args.length==0){
                ((Player) sender).sendMessage("Hola "+((Player) sender).getName()+"!!!");
            }
        }
    }
    return true;
}
```

Para gestionar correctamente el uso de comandos hay varios conceptos que se deben tener en cuenta:

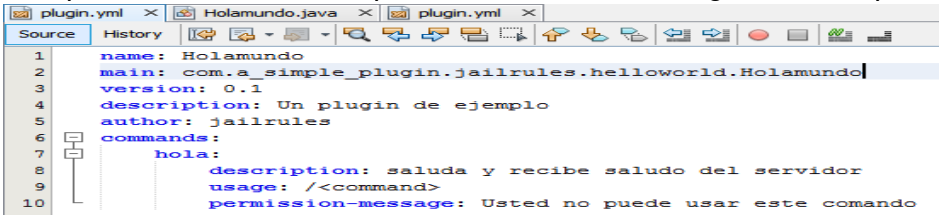
La procedencia del comando puede ser de consola o de un jugador, por ello se debe hacer una comparación si el comando así lo requiere La etiqueta del comando es la palabra introducida directamente tras la barra en el chat, si no existe mandará un mensaje de error. El número de argumentos que un comando puede tener puede ser variable, por lo que es necesario comprobar tanto el número de argumentos, cómo si los propios argumentos estan correctamente escritos.
Por último se debe crear en el paquete por defecto, o raiz del complemento, un fichero denominado plugin de extension yml



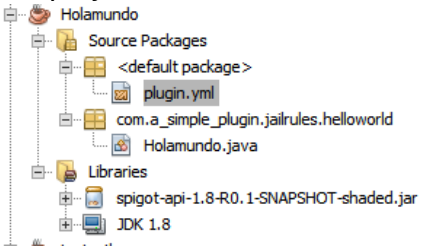
Este archivo contiene la dirección en la que el servidor debe encontrar la clase principal del complemento, el nombre de la clase en cuestión, el nombre del autor, una descripción, la versión y los comandos.



Completar los campos es el siguiente paso:



El proyecto en NetBeans debería verse de la siguiente manera:

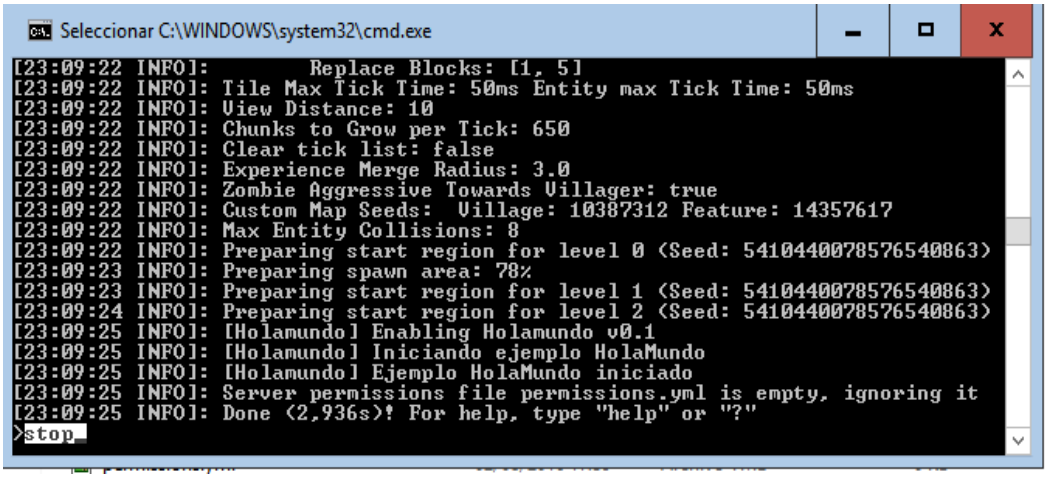


- 7º Por último comprobar el complemento en un servidor Spigot.

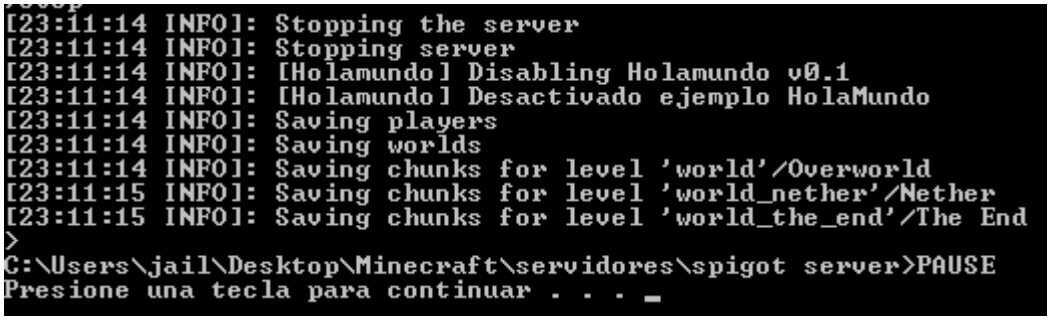
5. Instalación de complementos

El proceso de instalación y configuración de un complemento se realiza con los siguientes pasos, se hará uso del mismo ejemplo del punto anterior (Holamundo.jar):

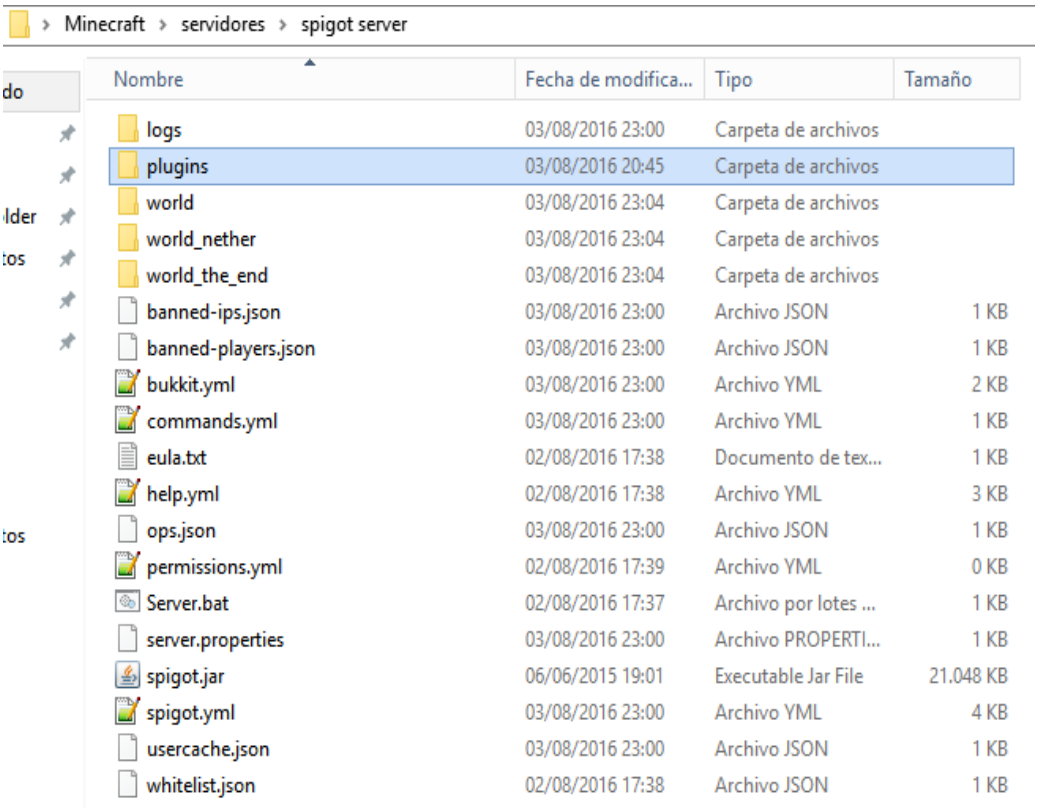
- 1º Para la instalación del complemento es necesario que el servidor este inactivo, por lo que se deberá parar si no es el caso. Si el servidor es remoto usar las herramientas propias del servidor de hospedaje, si el servidor es local desde la consola escribir “stop”, pulsar la tecla “enter” y esperar a que el servidor esté parado completamente.



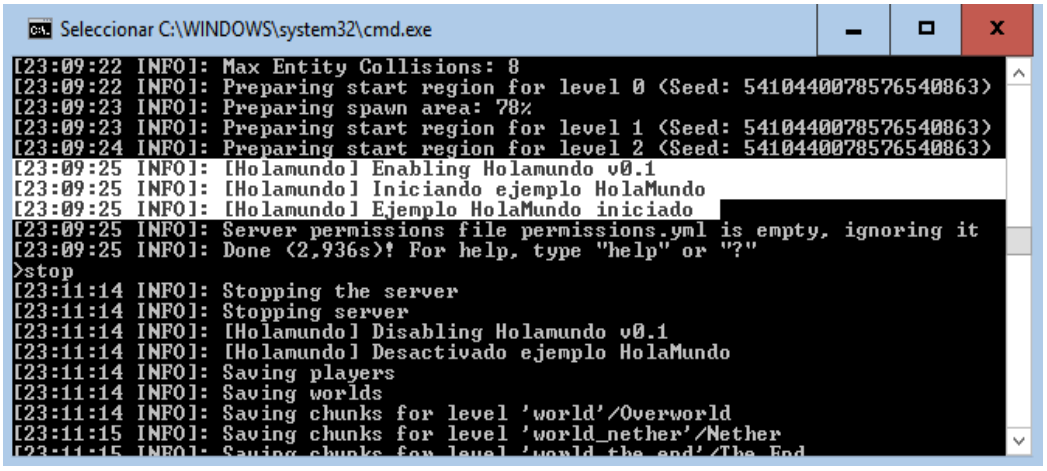
Una vez parado se mostrará así:



- 2º Mover el complemento a la carpeta plugins del servidor. Si el servidor es remoto usar las herramientas del servidor de hospedaje.



- 3º Iniciar el servidor y comprobar que en la consola se ha iniciado correctamente el complemento. En el ejemplo Holamundo.jar en el método de inicio del complemento se añadieron unos mensajes que indican si se ha cargado correctamente.



- 4º Iniciar el servidor por el cual se creará la carpeta por defecto del complemento y dentro sus archivos de configuración Yaml. En el ejemplo no se hacen uso de archivos de configuración por lo que no se creará nada.
- 5º Si el complemento no tiene la posibilidad de ser configurado dentro del juego, entonces será necesario parar el servidor, modificar los archivos de configuración y volver a iniciarlo.

12 Glosario de términos

C

clan
: Agrupación de personas unidas con un mismo fin y lideradas por una persona. En el entorno de los videojuegos de rol son como gremios que luchan por ser los mejores de un servidor. 16, 18, 23, 27, 28, 29, 65, 67, 68, 69, 87, 88

clase
: Una clase o profesión en los videojuegos de rol, caracteriza al personaje del jugador que la elige, modificando de manera distintas los atributos y habilidades que es capaz de realizar. 16, 17, 24, 25, 34, 35, 36, 44, 54, 63, 64, 65, 69, 71, 72, 73, 74, 77, 80, 83, 84, 85, 86, 87, 88, 90, 93, 94, 96, 97, 105, 124, 125, 126, 127

Ch

chunk
: porción de mapa de 16 de ancho, 16 de largo y 256 de alto en bloques que representa como son guardados y cargados en memoria del servidor. 78, 115

G

grupo
: En el entorno de los videojuegos de rol son agrupaciones de carácter temporal limitado, que comparten los bienes obtenidos de sus cruzadas. 16, 18, 29, 30, 65, 67, 68, 69, 70, 88

H

HotBar
: Barra de acceso rápido al inventario del jugador. En Minecraft son 9 huecos que contienen objetos que se pueden usar sin necesidad de abrir el inventario. 72, 73

M

MMORPG
: Los videojuegos de rol multijugador masivos en línea, son videojuegos de rol que permiten la conexión de un numero ilimitado de jugadores al servidor. Permiten crear un personaje, seleccionar una clase o profesión y que aumenten su nivel mediante la adquisición de experiencia y bienes, mejorando así los atributos de su clase. 1, 3, 7, 23, 54, 55, 57, 103, 104

mod
: Son modificaciones sobre los juegos que permiten la adición de nuevas funcionalidades. En Minecraft estas modificaciones requieren de una adaptación de la parte cliente. 10, 55, 117

monstruo
: En minecraft se refiere a entidades hostiles y neutrales que no son animales. En el proyecto se refiere a todos los tipos de entidades ya sean animales o monstruos. 9, 21, 33, 34, 50, 51, 52, 54, 66, 77, 78, 94, 95, 96, 97, 98, 99

P

pathfinder
: Medio por el cuál las entidades de Minecraft adquieren cierta inteligencia artificial. La estructura básica de un pathfinder consta de un método de guarda y un método de acción. Cuando la guarda se cumple realiza la acción. La agrupación de varios pathfinders en una entidad con sus respectivas prioridades crean una IA compleja. 78, 97

plugin
: O complemento, Al igual que las modificaciones o mods, los plugins añaden nuevas funcionalidades en el juego. En servidores no oficiales permiten que estas nuevas funcionalidades no requieran de una modificación de la parte cliente para su uso. 3, 54, 121, 122, 124, 126, 127

PNJ
: Personaje que no es un jugador. Hace referencia a otras entidades "vivientes" que se manifiestan en el juego, en este caso los monstruos. 109

S

spawner
: En Minecraft es un punto de aparición de un mismo tipo de monstruos representado mediante un cubo con forma de jaula. En el proyecto es el punto de aparición de monstruos sin necesidad de un bloque. 21, 34, 51, 52