



Sistemas Informáticos

Curso 2003-2004

AVENTURA GRÁFICA MULTIJUGADOR

Jorge Koronis Pérez
Sergio Ordoño Marín
Elsa Yáñez Morante

Dirigido por:
Prof. Jorge Jesús Gómez Sanz
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Índice

RESUMEN.....	5
INTRODUCCIÓN AGM.....	6
1.INTRODUCCIÓN	7
ARQUITECTURA BÁSICA CON J2EE.....	8
DIAGRAMA DE ELEMENTOS J2EE DE NUESTRA APLICACIÓN	9
<i>Elementos de la arquitectura conJ2EE</i>	10
<i>Descripción textual de la comunicación entre los elementos anteriores durante las diferentes fases de ejecución del sistema.....</i>	10
ARQUITECTURA DEL LADO DEL CLIENTEJUGADOR	12
PAQUETE: <i>agm.ClienteJugador</i>	12
agm.clienteJugador.InterfazGrafica	12
agm.clienteJugador.ClienteJugador	13
ARQUITECTURA DEL LADO DEL SERVIDOR	15
PAQUETE: <i>agm.servidor.kernel</i>	15
agm.servidor.kernel.OyenteBean.....	15
agm.servidor.kernel.KernelBean	15
agm.servidor.kernel.GestorSistemaBean	15
PAQUETE: <i>agm.servidor.habitaciones.....</i>	15
agm.servidor.habitaciones.HabitacionBean	15
CONSIDERACIONES DE RENDIMIENTO:	16
DIAGRAMA DE CLASES Y PAQUETES	17
DIAGRAMAS DE SECUENCIA JUNTO CON SU FLUJO TEXTUAL EN TÉRMINOS DE LOS OBJETOS QUE INTERVIENEN.....	18
<i>Arranque del sistema.....</i>	18
Diagrama	18
Flujo de sucesos	18
<i>Crear personaje.....</i>	19
Diagramas	19
Flujo de sucesos	19
<i>Conectar personaje</i>	20
Diagrama	20
Flujo de sucesos.....	20
<i>Ejecutar acción.....</i>	21
Diagrama	21
Flujo de sucesos	22
<i>Envío mensaje de chat.....</i>	22
Diagrama	22
Flujo de sucesos	22
DISEÑO DE LA BASE DE DATOS.....	24
TABLAS.....	25
DEPENDENCIAS	25
CONSULTAS DE CREACIÓN	26
DISEÑO HISTORIAS AGM	28
1. EL SECRETO DEL PROFESOR.....	29
A. <i>Flujo de acontecimientos</i>	29
2. APROBADO POR OBRA Y GRACIA DEL “MAS ALLA”	31
A. <i>Flujo de acontecimientos</i>	31
DISEÑO OBJETOS NO PERSONAJE AGM.....	34
PASTILLERO	35
PASTILLA DE JABÓN.....	35
LLAVE.....	35
PUERTA A CONSERJERÍA	36
PUERTA A DESPACHO (DESPACHO CERRADO)	36
CAJÓN.....	36

PALANCA	36
GUANTES DE LÁTEX.....	37
BOTELLA DE CAVA	37
PUERTA A GARAJE	37
FÉMUR	38
MALETERO.....	38
PUERTA GENÉRICA	38
SERVILLETERO	39
SERVILETA	39
BOTE DE LÁPICES.....	39
LÁPIZ	39
MÁQUINA.....	40
PÓSTER	40
DISEÑO OBJETOS PERSONAJE NO JUGADOR AGM	41
CONSERJE	42
PROFESOR X	43
PROFESOR GERVÁS	45
FANTASMA.....	46
FERRETERO	48
ARQUITECTURA DETALLADA.....	50
1. ARQUITECTURA DEL SISTEMA	51
PAQUETE: <i>agm.ClienteJugador</i>	51
ClienteJugador:	51
PAQUETE: <i>agm.ClienteJugador.InterfazGrafica</i>	54
InterfazGrafica:	54
PAQUETE: <i>agm.ClienteAministrador</i>	57
ClienteAdministrador:	57
PAQUETE: <i>agm.objetos</i>	57
ObjetoComunicacion:	57
ObjetoRefresco:	58
NotificacionTextual:	59
MeterInv:	59
SacarInv:	60
MetePJ:	60
SacaPJ:	61
CambioEstado:	61
ObjetoActualizacion:	62
<i>agm.objetos.objetosNoPersonaje</i>	62
ObjetoNoPersonajeClase:	62
Jabon:	63
Llave:	63
Aprobado:	64
Pastillero:	64
Puerta:	64
PuertaADespacho:	65
PuertaAConserjeria:	65
Acciones:	66
<i>agm.objetos.personajesNoJugadores</i>	67
PersonajeNoJugadorClase:	67
Ferretero:	68
Conserje:	68
ProfesorX:	68
Frase:	69
<i>agm.objetos.personajesJugadores</i>	69
PersonajeJugador:	69
PAQUETE: AGM.SERVIDOR	71
<i>agm.servidor.habitaciones</i>	71
HabitacionBean:	71

Habitacion:	74
HabitacionBMP:	75
HabitacionDAO:	77
HabitacionHome:	78
HabitacionLocal:	78
HabitacionLocalHome:	80
HabitacionPK:	80
HabitacionUtil:	81
HabitacionData:	81
<i>agm.servidor.kernel</i> :	83
OyenteBean:	83
GestorSistemaBean:	83
GestorSistema:	85
GestorSistemaHome:	85
GestorSistemaLocal:	85
GestorSistemaLocalHome:	85
GestorSistemaSession:	86
GestorSistemaDAO:	86
GestorSistemaUtil:	87
KernelBean:	87
Kernel:	88
KernelHome:	88
KernelLocal:	89
KernelLocalHome:	89
KernelSession:	89
KernelDAO:	90
KernelUtil:	90
PAQUETE: AGM.DAO	91
KernelDaoImpl:	91
GestorSistemaDaoImpl:	91
HabitacionDaoImpl:	92
PRUEBAS	94
1.DISEÑO DE LAS PRUEBAS REALIZADAS	95
HISTORIAS DE ANÁLISIS	97
SISTEMA TRANSMISOR DE CONOCIMIENTOS	98
<i>Personajes</i>	98
<i>Objetos</i> :	98
<i>Flujo de sucesos</i>	98
ELECTROCUCIÓN EN EL PANEL DE BOTONES DEL ASCENSOR	99
<i>Personajes</i>	99
PJ's:	99
PNJ's:	99
<i>Flujo de sucesos</i>	100
CHANTAJE A UN PROFESOR DEL SIP ROBANDO UN EXAMEN PARA LUEGO CONSEGUIR DINERO	101
<i>Personajes</i>	101
PJ's:	101
PNJ's:	101
DESCUBRIR EL PASADIZO QUE LLEVA A LA FACULTAD DE DERECHO	103
<i>Personajes</i>	103
PJ's:	103
PNJ's:	103
<i>Flujo de sucesos</i>	103
APROBADO DE MÁXIMO RIESGO	105
UNAS BOTELLITAS... Y UN JAMÓN	105
¿AQUELLOS MARAVILLOSOS AÑOS?	106
REUNIÓN DE PROFESORES	107
SOLUCIONES A PROBLEMAS CON LOMBOZ	108

COMO INTEGRAR CÓDIGO DE DISTINTOS PROYECTOS LOMBOZ.....	109
SOLUCIÓN A CUANDO TRAS INTEGRAR Y EJECUTAR EJB GENERATE, XDOCLET MUESTRA XDOCLET CLASSPATH MISSING J2EE CLASSES	109
COMO CONFIGURAR MYSQL PARA EJECUTAR EL CLIENTE DEL CAPÍTULO 3 DEL TUTORIAL DEL LOMBOZ:	110
SOLUCIÓN AL PROBLEMA: CUANDO HACEMOS GENERATE EJB CLASSES DE BEANS CON TAGS DE XDOCLET @EJB . DAO CLASS, NO SE GENERAN LOS INTERFACES DAO CORRESPONDIENTES.	110
SOLUCIÓN AL PROBLEMA: CUANDO SE HACE UN BUILD, DIRÁ QUE FALTAN LOS SIGUIENTES ARCHIVOS .JAR ..	111
SOLUCIÓN AL ERROR QUE SALE AL HACER EL DEPLOY DEL BEAN DEL CAPÍTULO 3 DEL TUTORIAL DEL LOMBOZ:	111
SOLUCIÓN A BUGS EN LA GENERACIÓN DEL JBOSS.XML POR EJB GENERATE:	111
SOLUCIÓN AL PROBLEMA DE EJB NOT BOUND DEL BMP DEL CAPÍTULO 5 DEL TUTORIAL.	113
SOLUCION PARA QUE CUANDO INCLUYAMOS MÉTODOS EJB CREATE EN LOS FICHEROS DEL DIRECTORIO SRC, AUTOMÁTICAMENTE SE INCLUYA UN MÉTODO CREATE ASOCIADO EN LA INTERFAZ HOME.	114
SOLUCIÓN AL PROBLEMA DE QUE EL LOMBOZ IGNORA LOS BREAKPOINTS:	114
SOLUCIÓN PARA EVITAR ERRORES DE MEMORIA DE WINDOWS AL INTENTAR ABRIR UN PROYECTO:	115
CREA TU PROPIA HISTORIA TUTORIAL	116
1. QUE NECESITAS PARA CREAR TU HISTORIA.	117
2. RANGO DE ACCIONES POSIBLES.	117
3. CREANDO LOS OBJETOS.	118
<i>Objetos no personaje.</i>	118
<i>Objetos Personajes No Jugadores.</i>	120
4. MODIFICANDO EL MUNDO.	122
5. PREPARANDO LA HISTORIA.	123
BLIBLIOGRAFÍA:.....	125
PALABRAS CLAVE	125
AUTORIZACIÓN.....	126

Resumen

Este proyecto consiste en una aventura gráfica multijugador on-line basado en una historia en la facultad de informática de la UCM. El diseño del juego está basado en Java Beans Enterprise de J2EE para la arquitectura cliente servidor y en componentes swing de J2SE para la interfaz gráfica. Hemos usado el entorno Eclipse como plataforma de desarrollo. Además hemos aprovechado las siguientes herramientas que vienen integradas en el entorno del servidor JBOSS, Xdoclet, Ant y el plugin Lombok.

This project is an on line multi user graphic adventure. It tells a history which happen on UCM Facultad de Informatica. The game design uses Java Bean Enterprise for server-client architecture and J2SE swing for graphic interface. We have worked on Eclipse environment which comes with JBOSS server, Xdoclet, Ant and Lombok plugin.

Introducción AGM

1.Introducción

La aplicación de la que vamos a hablar consiste en un videojuego de las llamados “aventuras gráficas” con la característica especial de ser multijugador online.

La idea básica es la de coexistir en un mundo virtual con otros jugadores, interactuando con el medio mediante un número limitado de acciones con la intención de lograr unos objetivos, en este caso, se trata de conseguir aprobados de maneras poco ortodoxas dentro de una universidad complutense de informática virtual. La característica que lo hace especial con respecto a otras aventuras gráficas, además de la mencionada capacidad multijugador, es la necesidad de cooperación entre diversos personajes jugadores para la obtención de diversos objetivos, lo que obligará a estos a comunicarse abiertamente utilizando el chat del que dispone la aplicación.

Las características principales del sistema que hemos desarrollado son:

- Basado en tecnología J2EE. Esta tecnología facilita desarrollos distribuidos y el acceso a bases de datos y otros recursos, como servidores de correo o mensajería genérica.
- Persistencia de datos. Toda la información del juego se almacena en una base de datos relacional que soporta transacciones
- Extensibilidad. La arquitectura ha sido diseñada para facilitar la inclusión de nuevas aventuras, objetos o personajes.
Para más información sobre la escalabilidad mirar el documento adjunto “Crea tu propia historia.”
-

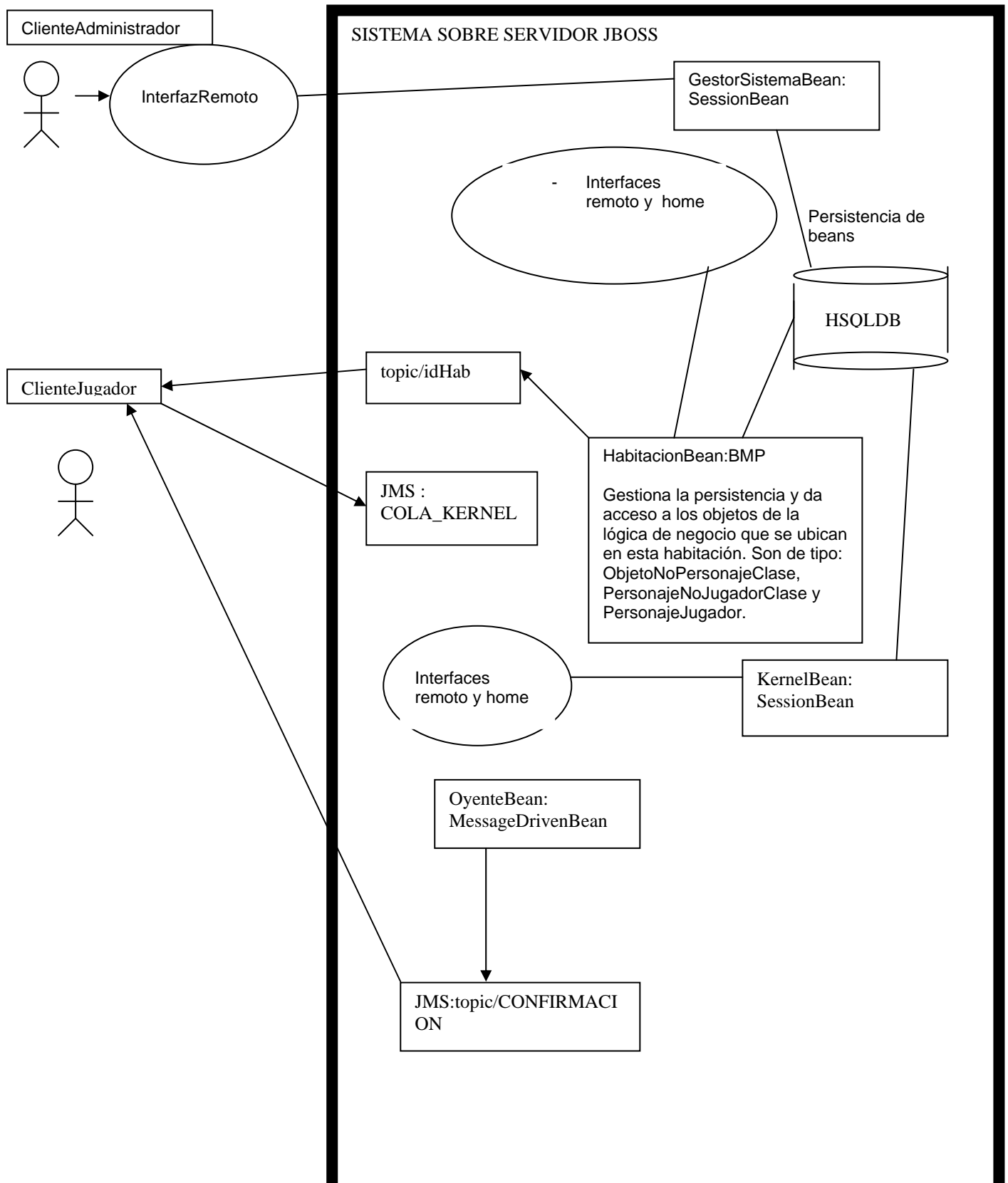
Otros aspectos:

- Plataforma de desarrollo. Eclipse con el plugin de Lomboz y con el JBoss como servidor de aplicaciones J2EE para desplegar nuestros beans y comunicación mediante mensajes JMS.
Se trata de una plataforma bastante versátil a la vez que conflictiva, y prueba de ello es el documento con las soluciones a los errores del Lomboz que nos encontramos y que se incluye en esta documentación.
- Una de las experiencias que nos ha aportado este trabajo es: confirmación de la importancia de la ingeniería del software para una perfecta colaboración entre los miembros de un equipo, y una optimización del tiempo y los recursos. Desgraciadamente, debemos admitir nos ha llegado gradualmente, y no hemos podido evitar algunos problemas de implementaciones diferentes de la misma cosa por parte de diferentes miembros del grupo, o trabajos duplicados, etc... Por suerte, pudimos paliarlos a tiempo, y la experiencia se torno enriquecedora.
- Resultado. Una aplicación consistente, que a falta de detalles de optimización, tales como mejoras de rendimiento o de interfaz gráfica, dispone de una alta escalabilidad y mucha libertad de acción.

Pasemos ahora a detallar el contenido de esta documentación.

Arquitectura básica con j2ee

Diagrama de elementos J2EE de nuestra aplicación



Elementos de la arquitectura conJ2EE

1. ClienteAdministrador: aplicación que se lanza para solicitar la creación del mundo.
2. ClienteJugador: aplicación que se lanza para crear y conectar personajes jugadores.
3. Sistema:
 - a. Beans:
 - i. GestorSistemaBean: proporciona el servicio de creación del mundo a través de su interfaz remoto.
 - ii. HabitaciónBean: representa cada una de las habitaciones del mundo. Escucha peticiones a través de su interfaz remoto y envía por topic/idHab notificaciones de modificación de su estado a los jugadores que están suscritos a su cola topic/idHab. Los clientes suscritos a su cola son lo que tienen al personaje en dicha habitación.
 - iii. OyenteBean: escucha por queue/COLA_KERNEL las peticiones que le hacen los ClienteJugador para crear, conectar o desconectar personajes jugadores y se encarga de que sean servidas comunicándose con HabitaciónBean para meter (CONEXIÓN) o sacar(DESCONEXIÓN) personajes de habitaciones y con Kernel para hacer el login y de creación de personajes jugadores al OyenteBean.
 - iv. KernelBean: proporciona servicios de login y de creación de personajes jugadores al OyenteBean.
 - b. Queues
 - i. Queue/COLA_KERNEL: una cola por la que el OyenteBean recibe las peticiones de creación, conexión y desconexión de personajes jugadores que le hacen los ClienteJugador.
 - c. Topics
 - i. topic/CONFIRMACIÓN: es el topic usado cuando OyenteBean envía las respuestas a las solicitudes hechas por el queue/COLA_KERNEL al OyenteBean.
 - ii. topic/idHab: hay uno por cada uno de las habitaciones del juego y sirve para transmitir el estado de la habitación a los ClienteJugador.
4. Gestor base de datos Hypersonic SQL Data Base(HSQLDB). Es el gestor de bases de datos que hemos usado para gestionar la persistencia.

Descripción textual de la comunicación entre los elementos anteriores durante las diferentes fases de ejecución del sistema

1. Puesta en marcha del juego. Se ejecuta la aplicación clienteAdministrador. Dicha aplicación invoca un método remoto de GestorSistemaBean para solicitar la creación del mundo. El GestorSistemaBean a su vez se comunica con el interfaz home de HabitaciónBean para crear las sucesivas habitaciones del mundo.

2. Creación de personaje jugador. Se ejecuta la aplicación ClienteJugador. Dicha aplicación solicita de manera asíncrona con Java Messaging Service (JMS) la creación de un personaje jugador al OyenteBean. El ClienteJugador envía un mensaje de solicitud de creación de personaje jugador a COLA_KERNEL. OyenteBean recibe dicha petición. El OyenteBean invoca un servicio de KernelBean para crear el personaje y una vez creado responde con la confirmación de personaje creado por la cola topic/CONFIRMACIÓN.
3. Conexión de personaje jugador. Se ejecuta la aplicación ClienteJugador. Dicha aplicación solicita de manera asíncrona con Java Messaging Service (JMS) la conexión de un personaje jugador al OyenteBean. El ClienteJugador envía un mensaje de solicitud de conexión de personaje jugador a COLA_KERNEL. OyenteBean recibe dicha petición. El OyenteBean invoca un servicio de KernelBean para crear el personaje y una vez creado responde la confirmación de personaje creado por la cola topic/CONFIRMACIÓN.
4. El ClienteJugador recibe la invocación del método ejecutaAccion desde la interfaz gráfica. Entonces el ClienteJugador invoca el método ejecutaAccion de HabitaciónBean. Entre los resultados obtenidos por la ejecución del método ejecutaAccion de HabitaciónBean quizá haya alguno que sea preciso enviárselo al cliente jugador. Para ello se usará el topic "topic/idHab"

Aquitectura del lado del ClienteJugador

PAQUETE: *agm.ClienteJugador*

agm.clienteJugador.InterfazGrafica

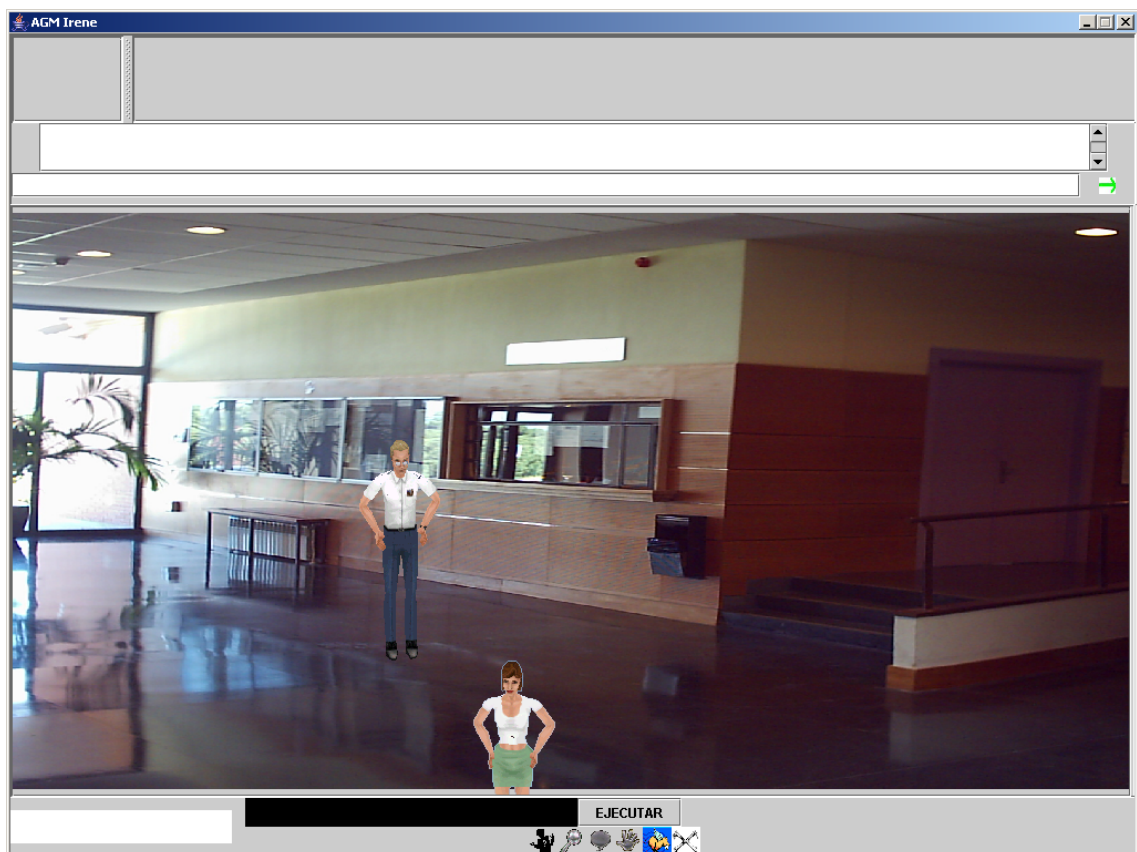
Consideraciones iniciales:

Suponemos que los jpgs que se usen para elegir el personaje cuando se cree una nueva partida fueron almacenados por el programa instalador de la aplicación cliente. En la última versión de la implementación no nos ha dado tiempo ha tener dicho instalador.

Para simplificar el interfaz gráfico se va a realizar una aproximación inicial donde:

- Tanto los personajes no jugadores como los jugadores no se mueven.
- Si un personaje jugador entra a una habitación, la posición en que se visualizan los personajes que ya estaban en la habitación es en la posición en que estaban cuando el jugador entra. Una vez el jugador ha entrado, cuando otro personaje de los que estaban dentro de la habitación, sale de ella, simplemente se muestra una desaparición del personaje que sale de la habitación. Y si un personaje nuevo entra en la habitación, la posición en que se va a colocar dicho jugador es aleatoria y dicha posición es la que van a ver los demás.

Captura de interfaz gráfica:



La pantalla del juego es un JPanel del api jdk. Sobre el JPanel se dibuja la imagen del fondo de cada habitación.

El inventario es un JSplitPane. Las 2 ventanas de chat son: una con JTextArea para los mensajes entrantes y otra de JTextField para los mensajes salientes.

La clase InterfazGrafica tiene clases privadas de oyentes de los diferentes eventos que pueden ocurrir sobre los componentes de la pantalla.

agm.clienteJugador.ClienteJugador

Es una clase de java que escucha en una cola Topic con nombre el "topic/idHab" donde idHab es el id de la habitación en la que se encuentra el personaje jugador. Además cuando se solicita creación de personaje jugador o conexión de personaje jugador la recepción de mensajes en el cliente jugador se hace por la cola cuyo nombre es "topic/CONFIRMACIÓN".

Esta clase cachea parte del estado de la habitación del intérprete y así reduce el tráfico de red. Consultando el atributo TipoObjeto de cada *ObjetosEnPantalla* con el que se quiere interactuar, se puede saber si queremos llevar a cabo acciones imposibles como hablar con puertas o cosas así. Si se detecta esa situación, se evita una llamada remota con el consiguiente ahorro en tiempo.

- Atributos:

Los siguientes atributos son una copia de los que posee la habitación. Habrá veces que puedan no estar actualizados durante un corto periodo de tiempo, hasta que reciba de la cola de la habitación en la que está escuchando una actualización de los mismos. A pesar de este inconveniente tenemos la ventaja de reducir el tráfico de red.

- private java.util.List personajesNJugadores;
- private java.util.List objetosNoPersonajes;
- private java.util.List personajesJugadores;

El cliente jugador se encarga de unir las tres listas de objetos de la lógica de negocio y generar a partir de ellos una lista de elementos de tipo ObjetoEnPantalla que será el tipo de los objetos manejados por la clase InterfazGrafica.

- Métodos:

- ClienteJugador

- public void desconectar(String idPJ)
Conecta con la cola del kernel y le hace una petición de desconexión. Cuando este la procesa y cuando recibe respuesta destruye la interfaz gráfica.
 - public void conectar()
Conecta con la cola del kernel, y le envía una solicitud de conexión con tu nombre de usuario y contraseña que previamente te pide en una ventana emergente, cuando recibe respuesta, esta va incluida con un objetoRefresco que usa para actualizar sus atributos y poder crear la interfaz gráfica.
 - public void creaPersonaje()
Crea una ventana de creación de personaje, donde introduces tus preferencias. Al aceptar, manda una petición de creación de personaje a la cola del kernel, el cual si no hay ningún problema, devolverá confirmación de registro.
 - OnMessage() – Este método recoge los mensajes de la cola a la que está conectado el ClienteJugador y procesa los mensajes que recibe (refrescando datos, mostrando texto de chat, etc...)
 - EjecutarAccion()
Este método es llamado por la interfaz gráfica. Se encarga de comprobar que la frase de acción generada por el jugador esté bien construida. Para ello comprueba que los objetos o las acciones no sean vacíos, e incluso consulta el tipo del objeto del parámetro de objetosEnPantalla para saber con que objeto de la lógica de negocio

se corresponde: ObjetoNoPersonajeClase,
PersonajeNoJugadorClase y PersonajeJugador.

Arquitectura del lado del servidor

PAQUETE: *agm.servidor.kernel*

La funcionalidad del kernel bean se reparte entre 3 beans: OyenteBean, GestorSistemaBean y KernelBean.

agm.servidor.kernel.OyenteBean

Escucha a través de la cola queue/COLA_KERNEL. Representa al Kernel del diseño solo en servicios **asíncronos** de recepción de solicitudes de creación y conexión de personajes jugadores. Después de recibir estas solicitudes, se sirven invocando los métodos de creación de personaje o de conexión del Kernel Bean.

Las colas de mensajes javax.jms.Topic tienen la ventaja de que se puede crear un subscriber *durable* (en terminología JMS API) a una de estas colas de mensajes. Si este subscriber está inactivo, cuando se vuelva a activar recibirá todos los mensajes que debieron ser entregados mientras estaba inactivo. Esto daría juego de cara a la recuperación de caídas en el sistema. En la última versión de implementación no está tratada la recuperación caídas de los clientes jugadores.

agm.servidor.kernel.KernelBean

Es un bean de sesión. Realiza el papel del Kernel central del diseño para servir las peticiones de creación y conexión de personaje jugador que le hace el kernel.

- Atributos:
 - private Collection PJConectados;
 - private Collection PJDesconectados;
- Métodos:
 - public boolean idLibre(String id)
 - public String creaPersonajeJ(String idPJ, String nombre, String password, String imagen) -- Es invocado por OyenteBean y crea un personaje nuevo con los datos suministrados en la petición que deja en la cola el clienteJugador de un Jugador.
 - ConsultarPJConectados o desconectados().
 - public boolean idLibre(String id)
Comprueba que la id no exista en el sistema.

agm.servidor.kernel.GestorSistemaBean

Es un bean de sesión que sirve las peticiones que le hace el clienteAdministrador de creación del mundo.

PAQUETE: *agm.servidor.habitaciones*

La funcionalidad del subkernel del diseño es implementada mediante el siguiente bean.

agm.servidor.habitaciones.HabitacionBean

Es un BMP. El interfaz remoto de este bean tiene un método ejecutAccion y uno de los parámetros de dicho método es el id de la acción el cual podrá tener los siguientes valores:

- "USAR"

- "HABLAR"
- "MIRAR"
- "IR A"
- "COGER"

Si el atributo `idAccion` toma el valor "IR_A" significa que el `ClienteJugador` ha detectado que se está intentando salir de la habitación y por ello para satisfacer ese servicio invoca el método `ejecutaAcción` de `Habitacion`.

Consideraciones de rendimiento:

En principio teníamos en mente hacer uso de las interfaces locales siempre que nos sea posible, es decir, siempre que las comunicaciones sean entre componentes que se ejecuten en la misma máquina pues así disminuiríamos la sobrecarga. El uso de interfaces remotas para comunicaciones dentro de la misma máquina es posible, pero es más costoso que el uso de los interfaces locales.

En la implementación final no usamos interfaces locales debido a que perdíamos mucho tiempo cambiando las clases generadas por XDoclet con el plugin Lomboz ya que dichas interfaces locales tenían bugs. El bug consistía en que en el fichero `jboss.xml` generado con EJB generate del plugin de Lomboz, ponía `ejb-ref` donde tenía que poner `ejb-local-ref`.

En este diagrama se puede ver la distribución de todas las clases de la aplicación en los distintos paquetes.

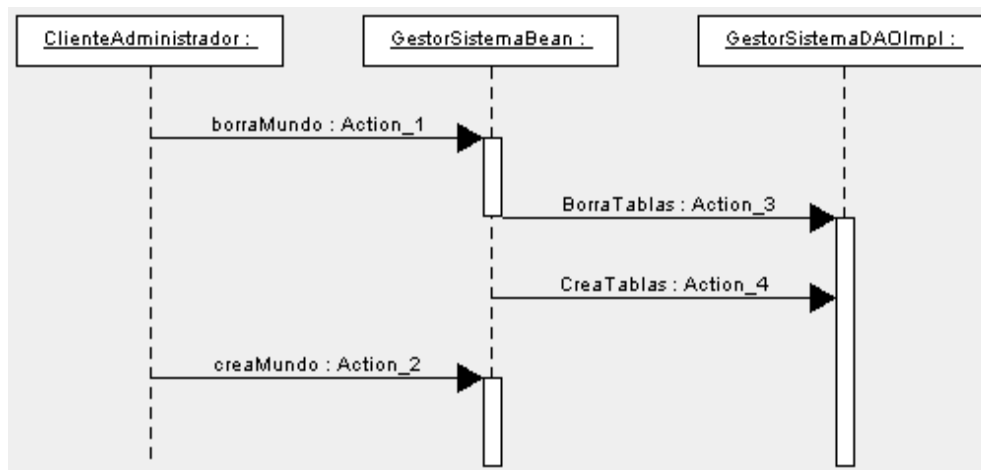


Diagramas de secuencia junto con su flujo textual en términos de los objetos que intervienen.

Los siguientes diagramas corresponden a los distintos casos de uso presentes en la aplicación.

Arranque del sistema

Diagrama

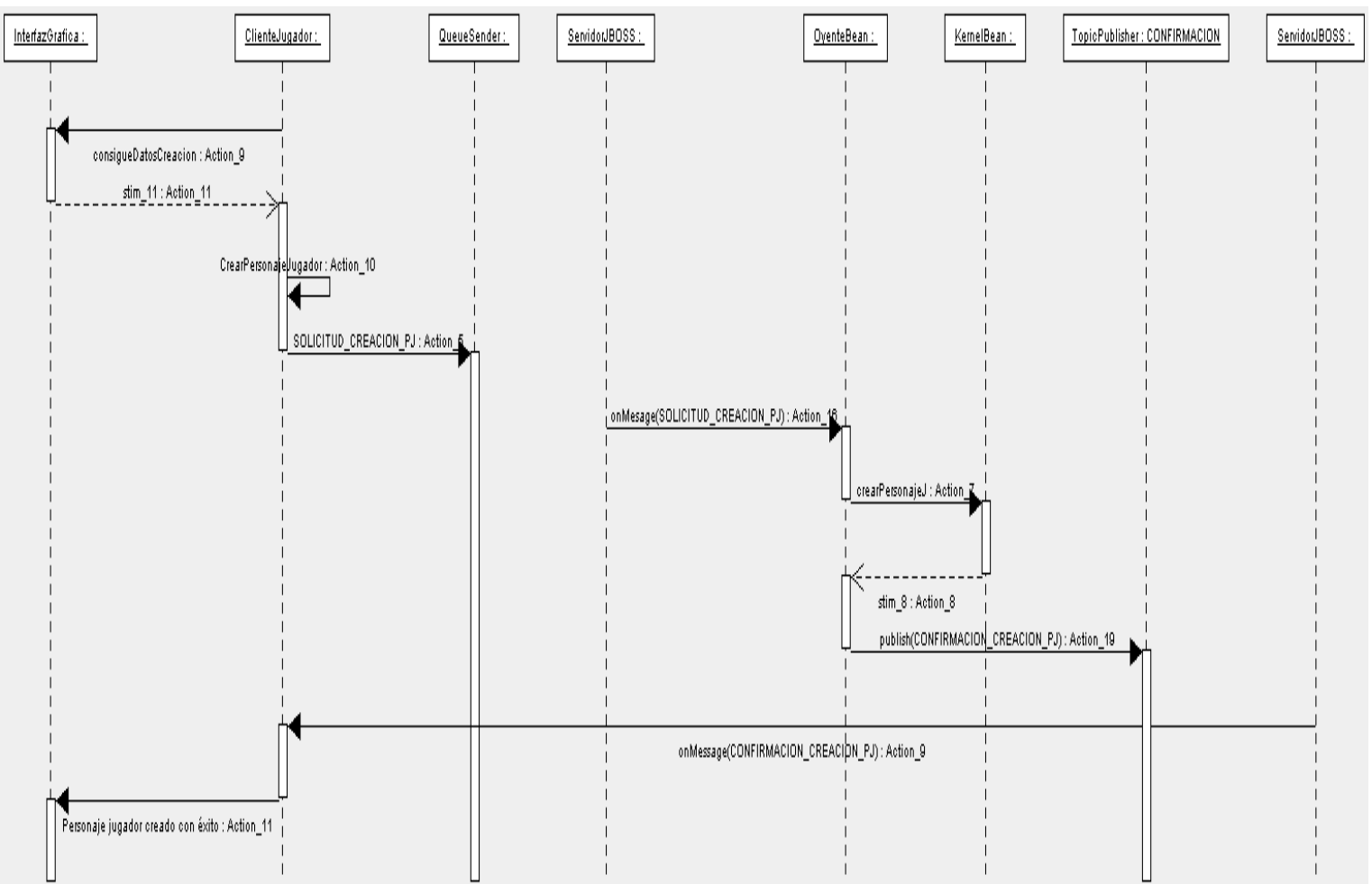


Flujo de sucesos

- ClienteAdministrador ejecuta una llamada **borraMundo()** sobre GestorSistemaBean
 - GestorSistemaBean ejecuta la llamada **borraTablas()** sobre GestorSistemaDaoImpl, el cual se encarga de acceder a la BD y borrar las tablas existentes, en caso de que ya estuvieran creadas.
 - GestorSistemaBean ejecuta la llamada **creaTablas()** sobre GestorSistemaDaoImpl, el cual se encarga de acceder a la BD y crear las tablas necesarias para el funcionamiento de la aplicación.
- ClienteAdministrador ejecuta una llamada **creaMundo()** sobre GestorSistemaBean. Este último crea las habitaciones, objetos y personajes no jugadores necesarios.

Crear personaje

Diagramas

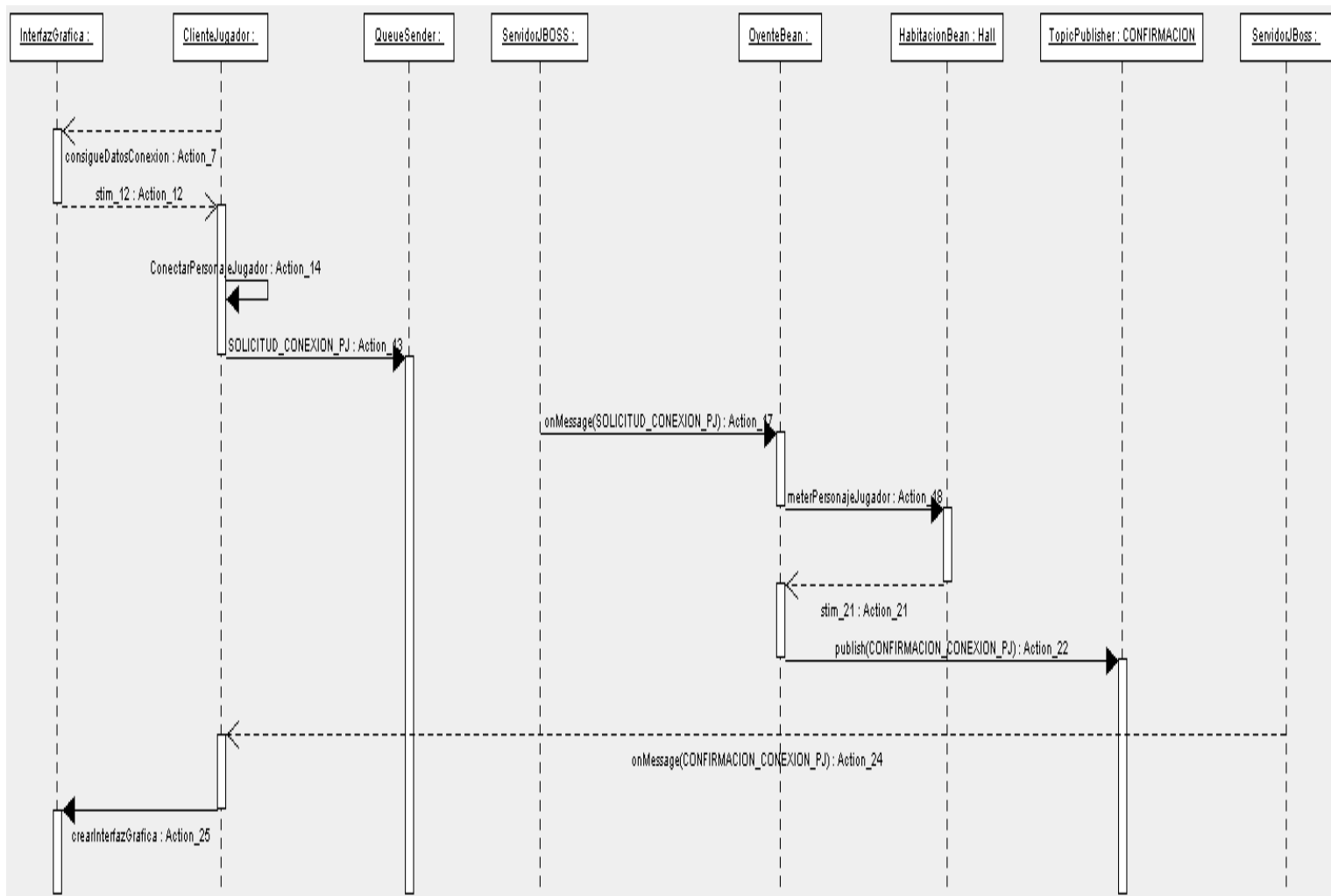


Flujo de sucesos

- El ClienteJugador hace una llamada a la InterfazGrafica para conseguir los datos necesarios.
- El ClienteJugador ejecuta sobre sí mismo la llamada creaPersonajeJugador
 - El ClienteJugador envía a la cola un mensaje del tipo SOLICITUD_CREACION_PJ.
- OyenteBean recibe el mensaje a través del servidor JBoss, y que procesa en el método OnMessage.
 - OyenteBean hace una llamada al método creaPersonajeJ de Kernel.
 - OyenteBean envía al topic un mensaje del tipo CONFIRMACION_CREACION_PJ.
- ClienteJugador recibe el mensaje a través del servidor JBoss y lo procesa en su método onMessage.
 - El ClienteJugador hace una llamada a la InterfazGrafica para mostrar un mensaje de éxito.

Conectar personaje

Diagrama



Flujo de sucesos

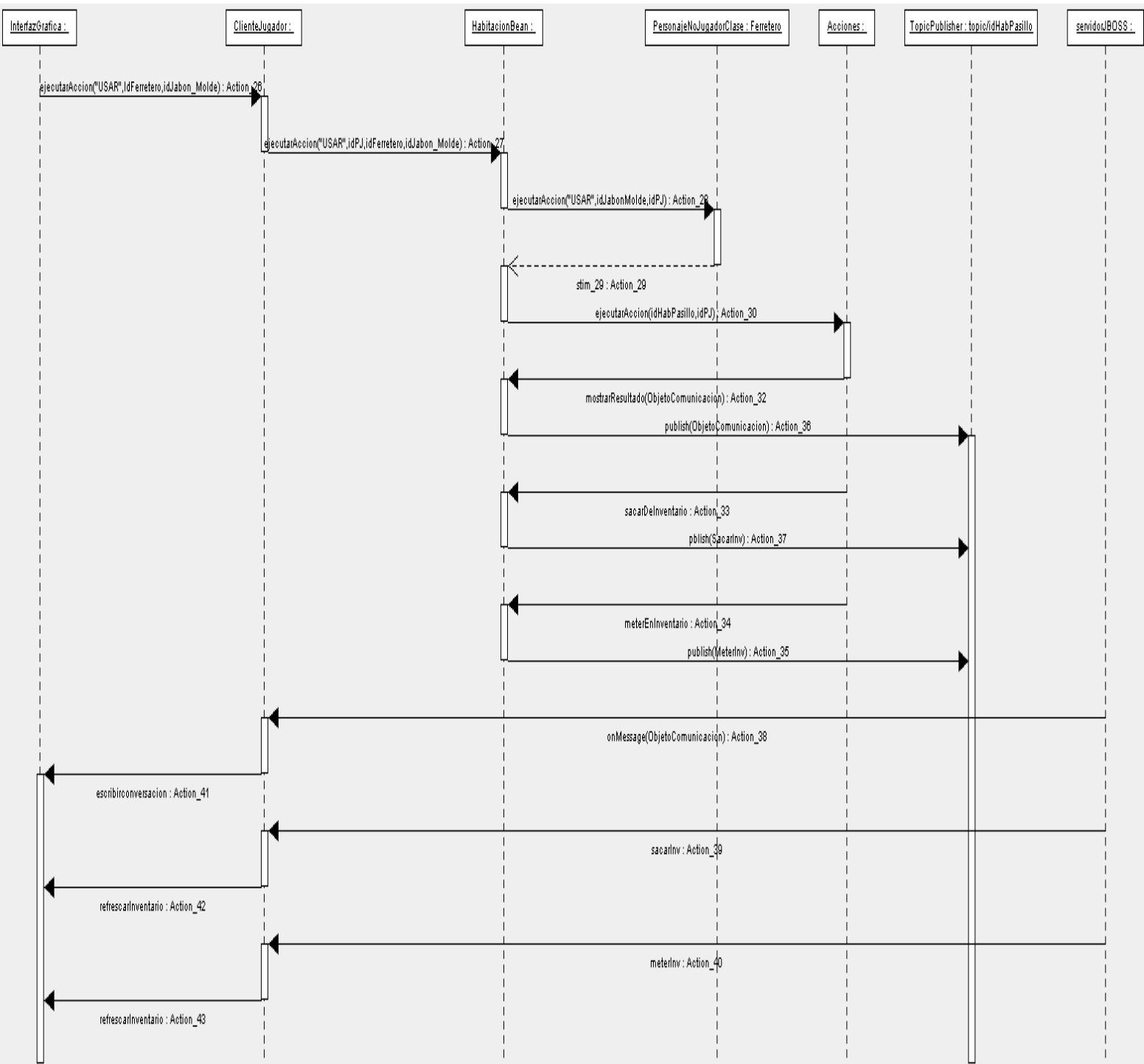
- El ClienteJugador hace una llamada a la InterfazGrafica para conseguir los datos necesarios.
- El ClienteJugador ejecuta sobre sí mismo la llamada conectarPersonajeJugador
 - El ClienteJugador envía a la cola un mensaje del tipo SOLICITUD_CONEXION_PJ.
- OyenteBean recibe el mensaje a través del servidor JBoss, y que procesa en el método OnMessage.
 - OyenteBean hace una llamada al método meterPersonajeJugador del HabitacionBean que representa al Hall.
 - OyenteBean envía al topic un mensaje del tipo CONFIRMACION_CONEXION_PJ.
- ClienteJugador recibe el mensaje a través del servidor JBoss y lo procesa en su método onMessage.

- El ClienteJugador hace una llamada a la InterfazGrafica para mostrar un mensaje de éxito.

Ejecutar acción

En este caso la acción a ejecutar será Usar Ferretero con jabón.

Diagrama

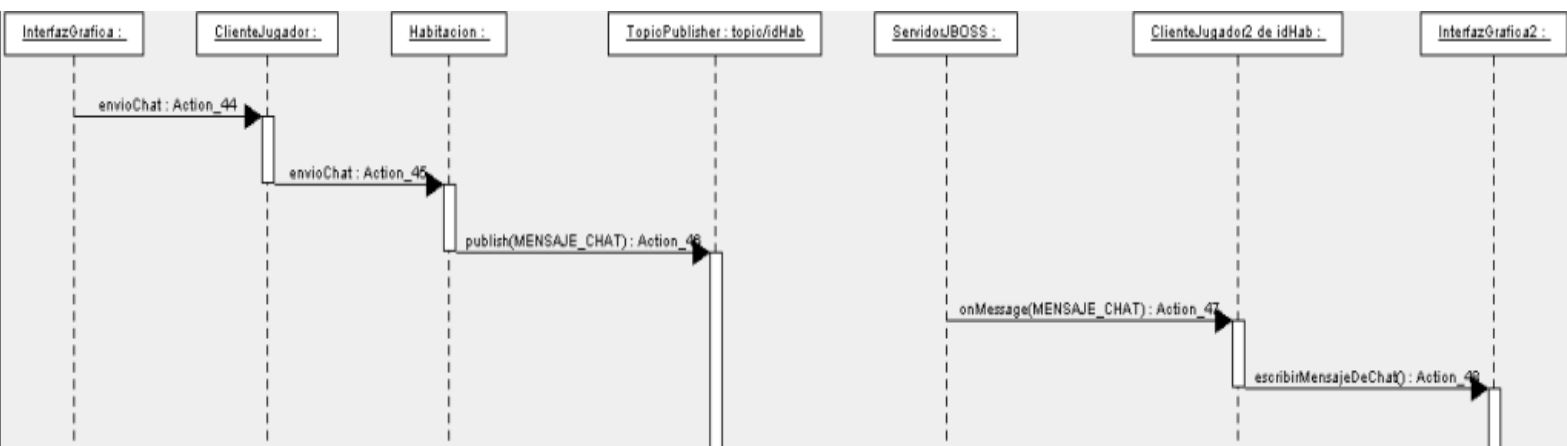


Flujo de sucesos

- La InterfazGrafica ejecuta ejecutarAccion(accion,objeto1,objeto2) sobre ClienteJugador. En este caso en particular la llamada será: ejecutarAccion("USAR", idFerretero,idJabon_Molde).
- El ClienteJugador ejecuta sobre el HabitacionBean correspondiente la llamada ejecutarAccion(accion,objeto1,objeto2).
 - HabitacionBean ejecuta la llamada ejecutarAccion(accion,objeto1,objeto2) sobre el objeto o personaje no jugador correspondiente, en este caso sobre Ferretero. Recibe un objeto de tipo Acciones.
 - HabitacionBean hace una llamada al método ejecutarAccion(idHab,idPJ) de Acciones.
 - Acciones realiza diferentes llamadas sobre métodos de HabitacionBean. En este caso son sacarDelInventario, meterEnInventario y mostrarResultado(ObjetoComunicacion).
 - HabitacionBean envía uno o varios mensajes al topic correspondiente.
- ClienteJugador recibe el mensaje a través del servidor JBoss y lo procesa en su método onMessage.
 - El ClienteJugador ejecuta sobre la InterfazGrafica la llamada correspondiente, que puede ser, entre otras, escribirConversacion o refrescarInventario, como en este caso en particular.

Envío mensaje de chat

Diagrama



Flujo de sucesos

- La InterfazGrafica ejecuta envioChat(mensaje) sobre ClienteJugador.

- El ClienteJugador ejecuta sobre el HabitacionBean correspondiente la llamada envioChat(mensaje,nombre).
 - HabitacionBean envía al topic correspondiente un mensaje del tipo MENSAJE_CHAT.
- El ClienteJugador de todos los personajes que se encuentran en esa habitación recibe el mensaje a través del servidor JBoss y lo procesa en su método onMessage.
 - ClienteJugador ejecuta sobre la InterfazGrafica la llamada escribirMensajeDeChat(nombre,mensaje).

Diseño de la base de datos

Tablas

PersonajesJ (IdP VARCHAR, Nombre VARCHAR, Password VARCHAR, Imagen VARCHAR, Dinero DECIMAL(7,2), IDHab VARCHAR, PosX INT, PosY INT)

Objetos (IdO VARCHAR, Estado VARCHAR)

SituacionO (IdO VARCHAR, IdHab VARCHAR, PosX INT, PosY INT)

Habitaciones(IdHab VARCHAR, Nombre VARCHAR, Luz BOOL, Ancho INT, Alto INT, Mascara CHAR, PosX INT, PosY INT)

Comunicacion (IdHab1 VARCHAR, IdHab2 VARCHAR, PosX INT, PosY INT, Cerrada BOOL, Tipo VARCHAR)

Inventario (IdP VARCHAR, IdO VARCHAR)

PersonajesNJ (IdPNJ VARCHAR, Nombre VARCHAR, Estado VARCHAR, IdHab VARCHAR, PosX INT, PosY INT)

Dependencias

SituacionO.IdO , Inventario.IdO , ObjetosHistoria.IdO → Objetos.IdO

Situacion.IdHab , Personajes.IdHab , PersonajesNJ.IdHab , Comunicación.IdHab1, Comunicación.IdHab2 → Habitaciones.IdHab

Inventario.IdP , PersonajeEstado.IdP → Personajes.IdP

Consultas de creación

```
create table Habitaciones(  
    IdHab Varchar(10) primary key,  
    Nombre Varchar(30) not null,  
    Luz tinyint not null,  
    Ancho Int not null,  
    Alto Int not null,  
    Mascara Varchar(150) not null  
);
```

```
create table PersonajesJ(  
    IdP Varchar(10) primary key,  
    Nombre Varchar(20) not null,  
    Password Varchar(10) not null,  
    Imagen Varchar(20) not null,  
    Dinero Decimal(7,2) not null,  
    IDHab Varchar(10) not null,  
    PosX Int not null,  
    PosY Int not null,  
    foreign key(IDHab) references Habitaciones(IDHab));
```

```
create table Objetos(  
    IdO Varchar(50) not null primary key ,  
    Tipo VarChar(20),  
    Estado Varchar(30));
```

```
create table SituacionO(  
    IdO Varchar(50) primary key,  
    IdHab Varchar(10) not null,  
    PosX Int not null,  
    PosY Int not null,  
    Tipo VarChar(20),  
    Estado Varchar(30),  
    foreign key(IdHab) references Habitaciones(IdHab),  
    foreign key(IdO) references Objetos(IdO)  
);
```

```
create table Comunicacion(  
    idO Varchar(50) not null,  
    IdHab1 Varchar(10) not null,  
    IdHab2 Varchar(10) not null,  
    primary key (idO),  
    foreign key(IdHab1) references Habitaciones(IdHab),  
    foreign key(IdHab2) references Habitaciones(IdHab));
```

```
create table Inventario(  
    IdP Varchar(10) not null ,  
    IdO Varchar(10) primary key ,  
    foreign key(IdP) references PersonajesJ(IdP),  
    foreign key (IdO) references Objetos(IdO)
```

);

```
create table PersonajesNJ(  
    IdPNJ Varchar(10) primary key,  
    Nombre Varchar(30) not null,  
    Estado Varchar(30) not null,  
    IdHab Varchar(10) not null ,  
    PosX Int not null,  
    PosY Int not null,  
    Tipo Varchar(20) not null,  
    foreign key(IdHab) references Habitaciones(IdHab));
```

DISEÑO HISTORIAS AGM

En detalle

1. El secreto del profesor

Objetos involucrados:

- Pastillero.
- Pastilla de Jabón.
- Llave del despacho.
- Puerta a Conserjería.
- Conserje (Objeto personaje no jugador).
- Puerta a despacho.
- Profesor (Objeto personaje no jugador).

Habitaciones involucradas:

- Hall
- Conserjería
- Baño
- Despacho 1
- Pasillo despachos

A. Flujo de acontecimientos

Precondición: La historia involucra a dos jugadores, que llamaremos A y B respectivamente.

El jugador A comienza en la habitación Baño, donde hay un Pastillero, el jugador B se encuentra en la habitación Hall.

- I. El jugador A realiza la acción “Coger Pastilla de Jabón” sobre el objeto pastillero.

La habitación invocará “EjecutarAcción(“Coger”,Vacio)” del objeto, y este devolverá:

- Valores de “**Resultado**” devuelto por el objeto Pastillero:

(“Esto puede serme útil... al menos para ir limpio por la vida”, True, False, [“METER_A_INVENT”], [“Jabon”]).

- II. El jugador B realiza la acción “Hablar Con Conserje” sobre el objeto Conserje.

La habitación invocará “EjecutarAcción(“Hablar”,Vacio)”, del objeto Conserje y este devolverá:

- Inicio de conversación. Ver documento Objetos Personajes no jugadores. Su estado cambia a “Conversando)

Precondición: El jugador A se desplaza hasta el Hall. El Jugador B continua conversando con el objeto personaje no jugador Conserje.

- III. El jugador A realiza la acción “Ir a Conserjería” sobre el objeto Puerta a Conserjería.

La habitación invocará “EjecutarAcción(“IrA”,Vacio)” del objeto, y este devolverá:

- Valores de “**Resultado**” devuelto por Puerta a Conserjería:

(“”, True, False, [“CAMBIO_HAB_CONDICIONAL”], [“conversando”, “Conserjería”]).

- IV. El jugador A realiza la acción “Usar Jabón con Llave”, siendo la llave uno de los objetos de la habitación Conserjería, y el jabón el suyo propio, adquirido en el punto I.

La habitación invocará “EjecutarAcción(“Usar”,“Llave”)” del objeto Jabón, y este devolverá:

- Valores de “**Resultado**” devuelto por Jabón:

(“¡Ey!, apretando la llave con fuerza sobre el jabón, puedo obtener un rudimentario molde de la misma, ¡genial!”, True, False, [“CAMBIAR_ESTADO_OBJ”], [“Jabón”, “Conserjería”]).

Nota: En caso de realizarse la acción “Usar Llave Con Jabón”, el valor de “**Resultado**” que devolvería la llave al invocar “EjecutarAcción(“Usar”,“Jabón”)” sobre ella sería:

*(“Eso no tiene sentido”, **False**, False, [], []).*

Precondición: El jugador A se desplaza hasta el pasillo de los Despachos tras haber convertido el molde en una copia de la llave en la tienda (se produciría un METE_A_INVENT, y SACA_DE_INVENT).

- V. El jugador A realiza la acción “Usar Llave con Puerta Cerrada” sobre el objeto Puerta a Despacho que hay en la habitación mencionada.

La habitación invocará “EjecutarAcción(“Usar”,“Llave”)” del objeto puerta a Despacho, y esta devolverá:

- Valores de “**Resultado**” devuelto por Puerta a Despacho:

(“”, True, False, [“CAMBIO_HAB”], [“Despacho”]).

Nota: Como en el caso anterior, Llave devolvería:

*(“Eso no tiene sentido”, **False**, False, [], []).*

Precondición: En la habitación Despacho se encuentra el objeto PNJ “Profesor”.

- VI. El jugador A realiza la acción “Hablar con Profesor” sobre el objeto Profesor.

La habitación invocará “EjecutarAcción(“Hablar”,Vacio)” del objeto profesor, y esta devolverá:

- Inicio de conversación. Ver documento Objetos Personajes no

jugadores. Su estado cambia a “Conversando”). Al final de la conversación, devolverá un “METER_A_INVENT” con “Aprobado”, un objeto que certifica que el jugador ha aprobado la asignatura del profesor.

2. Aprobado por obra y gracia del “mas alla”.

Objetos involucrados:

- Guantes de Látex.
- Cajón.
- Botella de Cava.
- Fémur.
- Maletero.
- Puerta a Garaje.
- Palanca.
- Profesor Gervás (Objeto personaje no jugador).
- Fantasma (Objeto personaje no jugador).

Habitaciones involucradas:

- Cafetería.
- Azotea.
- Pasillo Biblioteca.
- Despacho 2.
- Garaje.

A. Flujo de acontecimientos

Precondición: La historia involucra a dos jugadores, que llamaremos A y B respectivamente.

El jugador A comienza en la habitación Cafetería, donde está la puerta que lleva al garaje protegida por una alarma, mientras que el jugador B se encuentra en la habitación Despacho 2 (se accede a él desde pasillo a despachos). En dicha habitación hay un cajón y una grapadora como objetos.

- I. El jugador B realiza la acción “Usar Cajón” sobre el objeto Cajón, que la interfáz gráfica traducirá como “Abrir Cajón”.

La habitación invocará “EjecutarAcción(“Usar”,Vacio)” del objeto, y este devolverá:

- a. Valores de “**Resultado**” devuelto por el objeto Cajón:

(“Hecho.”, True, False, [“CAMBIAR_ESTADO_OBJ”], [“Cajón”]).

- II. El jugador B realiza la acción “Mirar Cajón” sobre el objeto Cajón.

La habitación invocará “EjecutarAcción(“Mirar”,Vacio)” del objeto, y este devolverá:

- a. Valores de “**Resultado**” devuelto por el objeto Cajón:

(“¡Cielos!... está lleno de cosas que... ¡Madre Mía!, unas bolas chinas, velas, un látigo de siete colas... ¿y esto?... ¡un montón de guantes de latex!, ¿para que demonios querrá nadie...? En fin, voy a coger unos guantes de estos, como hay tantos, nadie notará su ausencia... y mejor me marchó de aquí

antes de que venga el profesor que ocupa este despacho....”, True, False, [“METER_A_INV”, “CAMBIAR_ESTADO_OBJ”], [“Guantes”, “Cajón”]). –
Nota: El personaje cierra el cajón al coger los guantes.

- III. El jugador A realiza la acción “Hablar con Gervas” sobre el objeto PNJ Profesor Gervás.

La habitación invocará “EjecutarAcción(“Hablar”, Vacio)” del objeto, y este devolverá:

- a. Se producirá una conversación con el profesor, en la que si se desea y se dispone de dinero suficiente, se puede comprar una botella de cava, por lo que se producirá un: “METER_A_INVENT” con “Cava”.

Precondición: El jugador B se desplaza hasta la habitación “Azotea”, donde hay una aparatosa máquina de voltaje eléctrico con una palanca que pone: “Alarma ON, OFF”.

- IV. El jugador B realiza la acción “Usar Guantes con Palanca” usando los guantes de su inventario con el objeto “Palanca”.

La habitación invocará “EjecutarAcción(“Usar”, “Guantes”)” del objeto Palanca, y este devolverá:

- a. Valores de “**Resultado**” devuelto por el objeto Palanca:

(“Bien, con estos guantes esa palanca no me meterá ningún petardazo.”, True, False, [“CAMBIAR_ESTADO_OBJ_DX”], [“Puerta a Garaje”, “Pasillo a Biblioteca”]). **Nota:** Aquí usamos un comando no definido aún en la versión 3.5 del documento del diseño de la implementación, se trata de cambiar el estado de un objeto de otra habitación diferente, por eso este comando recibe dos parámetros, el objeto al que se le cambia el estado, y la habitación donde está.

Nota 2: En caso de que la habitación consulte primero a los guantes, lo que estos devuelven será de la siguiente forma:

(“No puedo hacer eso.”, False, False, [], []).

Precondición: El jugador A se desplaza hasta la habitación “Pasillo a Biblioteca”, donde hay una puerta que da acceso al garaje, pero está protegida con alarma. En ese momento, el Jugador B ejecuta el punto IV, por lo que durante un minuto, no habrá alarma, y el estado del objeto puerta a garaje será “desactivada”.

- V. El jugador A realiza la acción “Ir a Puerta Garaje”.

La habitación invocará “EjecutarAcción(“IrA”, Vacio)” del objeto Puerta a Garaje, y este devolverá:

- a. Valores de “**Resultado**” devuelto por el objeto Puerta a Garaje:

(“”, True, False, [“CAMBIAR_HAB”], [“Garaje”]).

- VI. El jugador A, una vez en el garaje, realiza la acción “Hablar con Fantasma” sobre el objeto PNJ fantasma de dicha habitación.

La habitación invocará “EjecutarAcción(“Hablar”, Vacio)” , y ocurrirá lo siguiente:

- a. Tras una conversación que se puede consultar en el documento de diseño

de PNJs, el jugador obtendrá información sobre los intereses del fantasma: alcohol.

- VII. El jugador A realiza la acción “Usar Botella de Cava con Fantasma”, la interfaz gráfica mostrará ese mensaje como “Dar Botella de Cava a Fantasma”.

La habitación invocará “EjecutarAcción(“Usar”, “Cava”)” del objeto PNJ Fantasma, y este devolverá:

- a. Tras una pequeña conversación el jugador recibirá el objeto “Fémur”, al enviar como resultado: [“SACAR_DE_INVET”, “Cava”, “METER_A_INVET”, “Fémur”].

- VIII. El jugador A realiza la acción “Usar Fémur con Maletero”.

La habitación invocará “EjecutarAcción(“IrA”, “Fémur”)” del objeto Maletero de la habitación Garaje, y este devolverá:

- a. Valores de “**Resultado**” devuelto por el objeto Maletero:

(“Eh!, genial, se abrió. Que mierda de coches hacen ahora, que no pueden resistirse a que lo fuerzen un poco con un buen fémur, jejeje... A ver que hay dentro... (¡Cojonudo!, el profe se ha dejado aquí las copias de los exámenes. Tiene tantas que seguro que no echará en falta una, jejeje. Y dejaré cerrado el maletero para que no sopeche nada, jejeje, ¡es perfecto!”), True, False, [“METER_A_INVENT”], [“Aprobado”]).

DISEÑO OBJETOS NO PERSONAJE AGM

Pastillero

Historias en las que está involucrado: "El secreto del profesor".

Parámetros:

- Nombre: "Pastilla de Jabón".
- Estados posibles: "" -- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Es un pastillero rebosante de pastillas de jabón", True, False, [], []).
- **Coger (por primera vez)**: ("Esto puede serme útil... al menos para ir limpio por la vida", True, False, ["METER_A_INVENT"], ["Jabon"]).
- **Coger (por segunda vez y sucesivas)**: ("No necesito más, con una me basta para conservar la higiene", True, False, [], []).
- **Fallo Acción ()**: ("No tiene sentido que haga eso", True, False, [], []).

Pastilla de Jabón

Historias en las que está involucrado: "El secreto del profesor".

Parámetros:

- Nombre: "Jabón".
- Estados posibles: ["Pastilla", "Molde llave"].

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar (Estado = "Pastilla")**: ("¡Uuuuum, que bien huele!", True, False, [], []).
- **Mirar (Estado = "Molde llave")**: ("Es la pastilla de jabón que recogí en el baño con la forma de la llave que había en conserjería. Un ferretero podría hacer maravillas con esto", True, False, [], []).
- **Usar (Objeto Destino = "Llave")**: ("¡Ey!, apretando la llave con fuerza sobre el jabón, puedo obtener un rudimentario molde de la misma, ¡genial!", True, False, ["CAMBIAR_ESTADO_OBJ"], ["Jabón", "Conserjería"]).
- **Fallo Acción ()**: ("Nah... Eso no serviría para nada", True, False, [], []).

Llave

Historias en las que está involucrado: "El secreto del profesor".

Parámetros:

- Nombre: "Llave despacho".
- Estados posibles: ["Original", "Copia"].

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar (Estado = "Original")**: ("Parece la llave de uno de los despachos de profesores.", True, False, [], []).
- **Mirar (Estado = "Copia")**: ("Es una copia de la llave de los despachos que hay en secretaría", True, False, [], []).
- **Coger ()**: ("Mejor que no haga eso... si el conserje me pilla, se me puede caer el pelo...", True, False, [], []).
- **Usar (Objeto Destino = "Jabón")**: ("Eso no tiene sentido", False, False, [], []).
- **Usar (Objeto Destino = "Puerta Despacho")**: ("Eso no tiene sentido", False, False, [], []).
- **Fallo Acción ()**: ("Nah... Eso no serviría para nada", True, False, [], []).

Puerta a Conserjería

Historias en las que está involucrado: "El secreto del profesor".

Parámetros:

- Nombre: "Puerta Conserjería".
- Estados posibles: [""].-- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (*"Es la puerta que lleva a conserjería. No sería conveniente que entre aquí, al menos mientras el conserje no esté distraído."*, True, False, [], []).
- **IrA ()**: ("", True, False, ["CAMBIO_HAB_CONDICIONAL"], ["conversando", "Conserjería"]).
- **Fallo Acción ()**: (*"...¿Qué?..."*, True, False, [], []).

Puerta a Despacho (Despacho Cerrado)

Historias en las que está involucrado: "El secreto del profesor".

Parámetros:

- Nombre: "Puerta Despacho".
- Estados posibles: [""].-- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (*"Parece que está cerrada."*, True, False, [], []).
- **Usar (Objeto Destino = "Llave")**: ("", True, False, ["CAMBIO_HAB"], ["Despacho"]).
- **Fallo Acción ()**: (*"No creo que eso diese resultado."*, True, False, [], []).

Cajón

Historias en las que está involucrado: "Aprobado por obra y gracia del 'mas allá'".

Parámetros:

- Nombre: "Cajón".
- Estados posibles: ["Cerrado", "Abierto"].

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar (Estado = "Cerrado")**: (*"Tiene una etiqueta que dice: Instrumentos necesarios para llevar a cabo revisiones de exámenes."*, True, False, [], []).
- **Mirar (Estado = "Abierto")**: (*"¡Cielos!... está lleno de cosas que... ¡Madre Mía!, unas bolas chinas, velas, un látigo de siete colas... ¿y esto?... ¡un montón de guantes de latex!, ¿para que demonios querrá nadie...? En fin, voy a coger unos guantes de estos, como hay tantos, nadie notará su ausencia... y mejor me marchó de aquí antes de que venga el profesor que ocupa este despacho..."*, True, False, ["METER_A_INV", "CAMBIAR_ESTADO_OBJ"], ["Guantes", "Cajón"]).
- **Usar (Estado = "Cerrado")**: (*"Hecho."*, True, False, ["CAMBIAR_ESTADO_OBJ"], ["Cajón"]).
- **Fallo Acción ()**: (*"No creo que eso diese algún resultado útil."*, True, False, [], []).

Palanca

Historias en las que está involucrado: "Aprobado por obra y gracia del 'mas allá'".

Parámetros:

- Nombre: "Palanca".
- Estados posibles: [""]. --- No usa ---

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (*"Es la palanca que controla la alarma de ciertas puertas de esta facultad. Si la pulso la alarma se apagará durante un rato, hasta que se active la corriente secundaria... pero no me fió nada de esta máquina, parece en muy mal estado."*, True, False, [], []).
- **Usar (Objeto Destino = "Guantes")**: (*"Bien, con estos guantes esa palanca no me meterá ningún petardazo."*, True, False, ["CAMBIAR_ESTADO_OBJ_DX"], ["Puerta a Garaje", "Pasillo a Biblioteca"]).
- **Usar (Objeto Destino = "")**: (*"¡No toco yo nada de esta máquina con las manos desnudas ni loco, ni borracho ni apaleao!"*, True, False, [], []).
- **Fallo Acción ()**: (*"No esperarás en serio que haga eso, ¿verdad?"*, True, False, [], []).

Guantes de Látex

Historias en las que está involucrado: "Aprobado por obra y gracia del 'mas allá'".

Parámetros:

- Nombre: "Guantes".
- Estados posibles: [""]. --- No usa ---

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (*"Son unos guantes de látex con una superficie extremadamente suave, y parecen impregnado en algún tipo de líquido lubricante"*, True, False, [], []).
- **Fallo Acción ()**: (*"No."*, True, False, [], []).

Botella de Cava

Historias en las que está involucrado: "Aprobado por obra y gracia del 'mas allá'".

Parámetros:

- Nombre: "Cava".
- Estados posibles: [""]. --- No usa ---

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (*"Es una botella de cava de marca 'QPoLLevo'. ¡Ummh!, rico y espumoso cava."*, True, False, [], []).
- **Fallo Acción ()**: (*"No esperarás en serio que haga eso, ¿verdad?"*, True, False, [], []).

Puerta a Garaje

Historias en las que está involucrado: "Aprobado por obra y gracia del 'mas allá'".

Parámetros:

- Nombre: "Puerta Garaje".
- Estados posibles: ["Activada, Desactivada"].

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Esta puerta lleva al garaje de la facultad, donde todos los profesores tienen sus cochazos, los muy jodíos. Por desgracia, está protegida con alarma", True, False, [], []).
- **IrA (Estado = "Desactivada")**: ("", True, False, ["CAMBIAR_HAB"], ["Garaje"]).
- **IrA (Estado = "Activada")**: ("No puedo pasar por aquí, la luz de la alarma está activada, me metería en problemas si la abro, True, False, [], []).
- **Fallo Acción ()**: ("Eso no tiene sentido.", True, False, [], []).

Fémur

Historias en las que está involucrado: "Aprobado por obra y gracia del 'mas allá'".

Parámetros:

- Nombre: "Fémur".
- Estados posibles: [""]. --- No usa ---

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("*Esto de aquí es un fémur de uno de los antiguos habitantes de la península, y parece muy duro y resistente, podría serme útil para muchas cosas*", True, False, [], []).
- **Fallo Acción ()**: ("Eso no tiene sentido.", True, False, [], []).

Maletero

Historias en las que está involucrado: "Aprobado por obra y gracia del 'mas allá'".

Parámetros:

- Nombre: "Maletero".
- Estados posibles: []. --- No usa ---

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("*Es el maletero del coche del profesor X. Arf, se me hace la boca agua pensando en que pueda tener los exámenes de Y ahí dentro. Si pudiera forzar la cerradura...*", True, False, [], []).
- **Usar (Objeto Destino = "Fémur")**: ("*Eh!, genial, se abrió. Que mierda de coches hacen ahora, que no pueden resistirse a que lo fuercen un poco con un buen fémur, je je je... A ver que tiene dentro... ¡Cojonudo!, el profe se ha dejado aquí las copias de los exámenes. Tiene tantas que seguro que no echará en falta una, je je je. Y dejaré cerrado el maletero para que no sospeche nada, je je je, ¡es perfecto!*", True, False, ["METER_A_INVENT"], ["Aprobado"]). --- Quitar Fémur si así se desea ---
- **Fallo Acción ()**: ("Eso no tiene sentido.", True, False, [], []).

Puerta Genérica

Historias en las que está involucrado: Ninguna.

Parámetros:

- Nombre: "Puerta a XXXXX".
- Estados posibles: []. --- No usa ---

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Esta puerta es el pasillo que lleva a XXXXX", True, False, [], []).
- **IrA ()**: ("", True, False, ["CAMBIAR_HAB"], ["XXXXX"]).

- **Fallo Acción ()**: (“Eso no tiene sentido.”, True, False, [], []).

Nota: Estas son las puertas que se usan para comunicar las habitaciones entre sí.

Servilletero

Historias en las que está involucrado: Ninguna.

Parámetros:

- Nombre: “Servilletero”.
- Estados posibles: [] -- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (“El típico servilletero de tasca de mala muerte, por suerte, lleno de servilletas”, True, False, [], []).
- **Coger (por primera vez)**: (“Voy a coger una, que nunca se sabe para que pueden venir bien”, True, False, [“METER_A_INVENT”], [“Servilleta”]).
- **Coger (por segunda vez y sucesivas)**: (“No necesito más servilletas”, True, False, [], []).
- **Fallo Acción ()**: (“No tiene sentido que haga eso”, True, False, [], []).

Servilleta

Historias en las que está involucrado: Ninguna.

Parámetros:

- Nombre: “Servilleta”.
- Estados posibles: [] -- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (“Una endeble servilleta de bar”, True, False, [], []).
- **Fallo Acción ()**: (“No puedo hacer nada de eso que dices”, True, False, [], []).

Bote de Lápices

Historias en las que está involucrado: Ninguna.

Parámetros:

- Nombre: “Bote”.
- Estados posibles: [] -- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (“Es un práctico bote de lapiceros. Hay muchísimos, ¿para qué querrá este profesor tantos lápices?”, True, False, [], []).
- **Coger (por primera vez)**: (“Si me llevo uno no se notará.”, True, False, [“METER_A_INVENT”], [“Lápiz”]).
- **Coger (por segunda vez y sucesivas)**: (“No quiero coger más”, True, False, [], []).
- **Fallo Acción ()**: (“Eso no servirá”, True, False, [], []).

Lápiz

Historias en las que está involucrado: Ninguna.

Parámetros:

- Nombre: "Lápiz".
- Estados posibles: [] -- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Es un lápiz tan afilado que me recuerda a... ¿una estaca?", True, False, [], []).
- **Fallo Acción ()**: ("Eso no servirá", True, False, [], []).

Máquina

Historias en las que está involucrado: Ninguna.

Parámetros:

- Nombre: "Máquina".
- Estados posibles: [] -- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Parece un generador de corriente en bastante mal estado", True, False, [], []).
- **Fallo Acción ()**: ("No pienso hacer eso", True, False, [], []).

Póster

Historias en las que está involucrado: Ninguna.

Parámetros:

- Nombre: "Póster".
- Estados posibles: [] -- No se usa --

Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Dice: THE TRUTH IS OUT THERE", True, False, [], []).
- **Fallo Acción ()**: ("No pienso hacer eso", True, False, [], []).

***DISEÑO OBJETOS
PERSONAJE NO
JUGADOR AGM***

Conserje

Historias en las que está involucrado: "El secreto del profesor".

Parámetros:

- Nombre: "Conserje".
- Estados posibles: "Conversando, Esperando"

I. Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Es uno de los 'amables' conserjes de la facultad, que 'tan bien' te facilitan información cuando la necesitas...", True, False, [], []).
- **Hablar (Estado = "Conversando")**: ("Está ocupado hablando con otro, tendré que esperar...", True, False, [], []).
- **Hablar (Estado = "Esperando")**: Se inicia una conversación. Ver punto II.
- **Fallo Acción ()**: ("No voy a hacer eso, me niego", True, False, [], []).

II. Frases conversación:

- Primera vez y única. (Este PNJ ofrece siempre la misma conversación, su finalidad es entretenerle y reírte un rato con sus respuestas).

1. Hola.

2. Hola, ¿en qué puedo ayudarte?.

3. Opciones:

3a. Quisiera una copia de mi expediente académico.

3b. Quiero información sobre convalidaciones de créditos de libre.

3c. ¿Que tal va todo por aquí?, ¿Alguna novedad?

3d. Nada, no es importante, ya vendré en otro momento.

4a. Aja... bien, mira, me rellenas este papelito, pagas estas altísimas tasas que te pongo aquí, vienes, dejas pasar un tiempo indeterminado el cual sea lo suficientemente grande como para que nos dé por trabajar y llevar a cabo tu solicitud, y entonces vienes y recoges tu expediente académico. Fácil, ¿no?

5a. Er... Vale... pero... es que lo necesito cuanto antes, no podríais...

6a. Estas usando la conjugación equivocada, chico, no sigas por ese camino, o lo lamentarás.

7a. Ya. Sí. Vale. No puede ser. Ok. Er...Lo cierto es que no esperaba otra respuesta. En fin, déjelo, ya me ocuparé de eso en otro momento.

8a. Muy bien, ¿deseas algo más?.

9a. Opciones: 3b, 3c, 3d.

4b. Bueno, depende de la modalidad y de con que convalides los créditos. Por ejemplo, si se trata de un curso de verano, tendrías que rellenar el formulario NCC-1701 seguido del NCC-1701-A. En caso de que además quieras pagar con pago fraccionado y devengando los derechos de cobro a la facultad a plazo fijo mensual tendrías que traerme una muestra de ADN, junto al formulario NX-01 debidamente cumplimentado. Y bueno, luego hay pequeños detalles, por ejemplo si eres Talaxiano deberás incluir también el formulario DS9, y así un largo etcétera. Pero bueno, no es muy complicado, no te preocupes.

5b. Opciones:

5b1. ¿NCC-1701?, ¿DS9?... ¿De que me suenan?.

5b2. ...¿Talaxiano?...

5b3. Esto... ¡Adios! --- Sin respuesta ---

6b1. Ni idea, para mi, no es más que jerga burocrática. Preguntale al alto mando de la facultad sobre el significado de las siglas, cadete. ¿O acaso crees que nosotros aqui lo sabemos todo?.

7b1. No, por supuesto que no. Este usted tranquilo, que eso no lo dudo en absoluto.

8b1. Bien, ¿se te ofrece alguna otra cosa?.

9b1. Opciones: 3c, 3d.

6b2. Bueno, no me malinterpretes, se ve a simple vista que tu no eres Talaxiano, pues no es difícil ni ná pasar por alto a uno de esos duendes de orejas puntiagudas sabelotodos, ¿no crees?.

7b2. Esto... claro, claro, eh... que jodios estos Talaxianos.

8b2. Sólo te lo decía como ejemplo, para que vieses como funciona el sistema. Los humanos no tienen que rellenar el formulario DS9, pero si él TNG.

9b2. Ya. Bueno, parece que son demasiadas cosas, ¿no?, mejor me ocupo de eso en otro momento.

10b2. Como quieras. ¿Se te ofrece alguna otra cosa?.

11b2. Opciones: 3c, 3d.

4c. Bueno, si quieres saber eso mira los tablonos o en la página web de la facultad. ¡Que siempre venis aquí a molestarnos preguntando cosas que ya están circulando por ahí!.

5c. Opciones:

5c1. Pero, para eso estais, ¿no?. Para informarnos. Vamos, digo yo.

5c2. Parece como si te molestara hacer tu trabajo.

6c1 y 6c2. Mira, no me toques... el temita ese, que no estoy de humor. ¿Quieres alguna cosa más, o me dejas seguir con mi buscamí... trabajo?.

7c1 y 7c2. Opciones: 3d.

4d. Adios.

Profesor X

Historias en las que está involucrado: "El secreto del profesor".

Parámetros:

- Nombre: "Profesor TAL".
- Estados posibles: "Conversando, Esperando"

I. Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Es TAL, el profesor de PASCUAL. Que aspecto tan rato tiene hoy...", True, False, [], []).
- **Hablar (Estado = "Conversando")**: ("Está ocupado hablando con otro, tendré que esperar...", True, False, [], []).
- **Hablar (Estado = "Esperando")**: Se inicia una conversación. Ver punto II.
- **Fallo Acción ()**: ("No voy a hacer eso, me niego", True, False, [], []).

II. Frases conversación:

- Primera vez que un PJ habla con él.

1. ¿Qué es esto?, ¿quién eres?, y lo más importante, ¿qué haces aquí en mi despacho?
2. Anda profesor TAL, esto... estaba la puerta abierta y...
3. ¿La puerta abierta?. CREI haberla cerrado.
4. No, no, ¡de verdad!, me la he encontrado abierta, no es que tenga una copia de la llave ni nada por el estilo...
5. Vale chico, lo que tu digas. Ahora, dime, quieres algo, ¿o puedes dejarme solo?
6. **Opciones:**
 - 6a. Quisiera que me aprobara su asignatura por su gracia divina.
 - 6b. Vengo a una tutoría.
 - 6c. Pues... ¿Porque hay tan poco luz aquí?. Trabaja usted casi en penumbra.
 - 6d. No, no es nada, lo siento, dejo de molestarle. --- Sin respuesta ---

7a. Anda, un cachondo, ¿eh?. Deberías saber que a mi no me gusta el trato informal con los alumnos, que yo soy un peso importante en esta facultad.

8a. No lo dudo, pero dígame, ¿en qué sentido es usted importante?.

9a. Pues por ejemplo, sin ir más lejos, yo soy el que dio el visto bueno al diseño de este edificio, que se dice pronto.

10a Opciones:

10a1. Aja. ¿Así que usted es quien eligió ese horrible color lila para las puertas?.

10a2. Anda, y dígame, ¿Por qué demonios orientó la puerta de la facultad de forma que nunca le dé el sol salvo en Agosto, lo cual provoca que se formen capas de hielo en invierno?.

11a1. Si bueno... todo tiene su razón de ser.

12a1. Lo eligió su mujer, ¿no es cierto?.

13a1. ¡Es que no se puede discutir con ella!.

14a1. Opciones: 10a2.

11a2. Pues verás... esto... ¿Es que acaso necesitas el sol para algo?. Es tan molesto, con su brillo cegador, y su manía de reducirte a cenizas si te descuidas un momento.

12a2: ¿Como?.

13a2. ¡Ops!, nada nada, no he dicho nada.

14a2. Opciones:

14a2a. ¿?... ahora que me doy cuenta, que largos tiene los colmillos, ¿no?, debería ir al dentista.

14a2b. Ejem... esta conversación no me está gustando nada, ya, vendré si eso en otro momento, ¡adiós!. --- Sin respuesta ---

15a2a. Eeeeeeh.... puesss... verás... esto... como te explicaría... ¿tu sabes guardar un secreto?.

16a2a. Si es a un buen precio, por supuesto.

17a2a. Ya me imaginaba que dirías algo así. Jodidas nuevas generaciones de listillos... ¿y que tal precio te parece un aprobado de mi asignatura?.

18a2a. ¡Me parece genial!, todo lo que se gana sin esfuerzo es muy bienvenido por mi parte. Lo aprendí de nuestros políticos, ¿sabe?.

19a2a. No me sorprende. En fin, si no cuentas por ahí nada de nuestras pequeña conversación, ni nada de lo que has visto aquí, te daré ese aprobado por escrito. ¿Hay trato?.

20a2a. ¡Por supuesto!.

21a2a. Estamos de acuerdo entonces. Voy a firmarte el aprobado, y te marchas de aquí, que empiezo a tener hambre, y debo controlarme.

22a2a -> se produce un METE_A_INVENT de "Aprobado X", y un CAMBIO_HAB a "Pasillo despachos".

7b. ¿Ahora?. Justo durante mi ritual... Esto... ¡Pero que inoportunos que llegáis a ser, humanos!, lo siento, pero ahora no tengo horario de tutorías, vuelve más tarde, y déjame sólo.

8b. Umm... si... cuando alguien me dice cosas raras como las tuyas, suelo marcharme, así que mejor vengo en otro momento.

9b. Adiós.

7c. Porque no soporto el maldito sol, podemos decir que es... perjudicial para mi piel.

8c. Creo que le entiendo. A mí también me molesta mucho cuando me dejo las Rabian en casa. De todas formas, podría dar usted la luz de la mesita, ¡qué parece un vampiro con tan poca luz aquí metido!.

9c. ¿Vam-vam-pi-ro?, ¡no no no no no por nosferatu!, me pueden acusar de chupar la sangre a mis alumnos, pero sólo lo hago metafóricamente. Ya me suple de sangre la cruz roja.

10c. Opciones: 14a2a y 14a2b

- Segunda vez que se habla con él, cuando ya se tiene el "aprobado X":

1. ¡Hola profesor TAL!.

2. ¿Tu por aquí otra vez?. ¡Márchate de aquí antes de que te arrepientas de haber venido mortal, y no olvides nuestro trato!.
3. Vale, vale, me marchó.

Profesor Gervás

Historias en las que está involucrado: "Aprobado por obra y gracia del más allá".

Parámetros:

- Nombre: "Profesor Gervás".
- Estados posibles: "Conversando, Esperando"

I. Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: ("Es Gervás, el profesor de IS. Que 'sorpresa' verle en la cafetería...", True, False, [], []).
- **Hablar (Estado = "Conversando")**: ("Está ocupado hablando con otro, tendré que esperar...", True, False, [], []).
- **Hablar (Estado = "Esperando")**: Se inicia una conversación. Ver punto II.
- **Fallo Acción ()**: ("No voy a hacer eso, me niego", True, False, [], []).

II. Frases conversación:

- Primera vez que un PJ habla con él.

1. Hola bambino, ¿cómo va todo?.
2. Bien, ¿y usted?.
3. Pues nada, aquí, haciendo ingeniería del software.
4. **Opciones**:
 - 4a. ¿Ingeniería del software?, ¿Aquí, en la cafetería, tomando unos lingotazos?.
 - 4b. Ah, claro, claro, que tonto, como no me había dado cuenta. ¿Y que tal se da?.
 - 4c. Bien, pues nada, que se dé bien, que me tengo que ir.

5a . Si claro, por supuesto, estoy haciendo conocimiento del medio, para capturar unos cuantos casos de uso.

6a. Pero... ¿de qué va la aplicación?.

7a. Calcula estadísticas de venta y consumo de alcohol en un bar o cafetería dado.

8a. Que... interesante. Y claro, por eso está aquí dándole al codo, ¿no?.

9a. Tú lo has dicho. Todo sea por la investigación y la ciencia.

10a. Claro, por supuesto, debería haberme dado cuenta antes. En fin, como que me voy a marchar, tengo cosas que hacer.

11a. Espera bambino, ¿no viniste aquí por la "cosa nostra"?.

12a. **Opciones**:

12a1. ¿Que es la "cosa nostra"?.

12a2. Estoooo... esta conversación no me está gustando nada, nos vemos en otro momento, ¿vale?.

Adiós. --- Termina conversación ---

5b. Bien, aquí estamos. Con la "cosa nostra".

6b. Bueno, pues me alegro que vaya todo bien.

7b. Dime una cosa, ¿no has venido a hablar conmigo por la "cosa nostra", bambino?.

8b. **Opciones**: 12a1 y 12a2

13a1. Pero bambino, ¿no sabes de lo que te estoy hablando?.

14a1. No.

15a1. Si quieres saberlo, puedo contártelo, y puede incluso que te interese, pero deberás prometerme que no se lo dirás a nadie, ¿capichí?.

Es tuto cuello lo que estaría en juego si me traicionaras, bambino.

16a1. **Opciones:**

16a1a. No me dejes ahora con la intriga, cuéntamelo, asumo las consecuencias.

16a1b. Creo que me están llamando. Nos vemos, ¿vale?. --- Fin conversación ---

17a1b. Eres valiente bambino. Escucha pues lo que te voy a decir. Yo sé lo mucho que los estudiantes necesitáis de unos buenos brindis para llevar mejor la dureza de estos estudios, ¿sabes?.

18 a1b. Oh, bueno, sí, unos más que otros, pero bueno, un trago no suele venir mal.

19 a1b. Claro que sí bambino. Y como desafortunadamente los camareros de esta cafetería han sufrido unos desagradables accidentes en los que yo no tengo nada que ver, no queda nadie para vender alcohol.

20 a1b. Vaya, pues supongo que eso es malo.

21 a1b. Malísimo, bambino. Por suerte, aquí estoy yo para cubrir vuestras necesidades.

22 a1b. Ah... que... ¿amable? por su parte...

23 a1b. No bambino, simplemente he usado los necesarios castigos que debo imponer en mi asignatura y que me cobro en forma de botellas de cava para suplir una necesidad de la facultad.

24 a1b. Así que estás repartiendo esas botellas de cava que te debe dar algún alumno cuando hace algo malo, ¿no es así?.

25 a1b. ¿Repartir?, no es esa la palabra correcta. Yo diría... 'regalar' a un módico precio. Pero basta de cháchara bambino, que tengo muchos asuntos que atender. No estarás interesado en una botellita, ¿verdad?. Tan sólo 100 euros por ser tu.

26 a1b. **Opciones:**

26a1b1. Bueno, es un poco cara, pero deme una, me sentará bien.

26a1b2. Ahora no me interesa, gracias, pero bueno es saberlo.

27a1b1. Estupendo, y tienes la guita y todo, como a mí me gusta. Aquí tienes tu botella bambino. Y recuerda, que siempre puedes venir a por más. **Resultado:** ([“QUITA_DINERO”, “METE_A_INV”], [100, “Cava”]).

27a1b2. Como quieras bambino, cuando la sed te apriete, ya sabes donde estoy. Sé que volverás.

28 a1b2. Adiós.

- Segunda vez y sucesivas que un PJ habla con él, cuando el jugador ya ha tenido opción a comprar botellas.

1. Hola otra vez bambino, vienes por la “cosa nostra”. Ya sabes, 100 euros y una botellita es tuya.

2. **Opciones:**

2a. Venga, dame una.

2b. Esperaba que hubiera bajado de precio. Sigue siendo muy cara, ya vendré luego.

3a. Estupendo, trae aquí esa guita, bambino, y toma tu botella. Tranquilo, sé que no te arrepentirás. ([“QUITA_DINERO”, “METE_A_INV”], [100, “Cava”]).

Fantasma

Historias en las que está involucrado: “Aprobado por obra y gracia del más allá”.

Parámetros:

- Nombre: “Fantasma”.
- Estados posibles: “Conversando, Esperando”

I. Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar (I):** (“¡Dios mío!, esto parece un fantasma de película de serie b.”, True, False, [], []).
- **Hablar (Estado = “Conversando”):** (“Parece que lo que sea esta cosa es capaz de hablar y ahora está ocupado hablando con otro, tendré que esperar...”, True, False, [], []).
- **Hablar (Estado = “Esperando”):** Se inicia una conversación. Ver punto II.
- **Usar (Objeto Destino = “Cava”):** Ver conversación 2.
- **Fallo Acción (I):** (“No”, True, False, [], []).

II. Frases conversación:

- Primera vez que un PJ habla con él.

1. Opciones:

1a. ¡Aaaaaah!.

1b. ...¿Qué eres?.

1c. Supongo que eres un fantasma o algo parecido, ¿verdad?. – Ir a 4b

1d. (Estas cosas me da muy mal rollo, mejor me marchó).

2a. Vaya que bien, me alegra saber que aún valgo para dar algún sustito que otro.

3a. ¡Aaaaaah!, ¡puedes hablar!.

4a. Bueno, realmente transmito mis pensamientos a tu mente mediante sonido de ultratumba camuflados, pero puedes considerarlo como hablar

5a. **Opciones:** 1b, 1c, 1d.

2b. Por tu aspecto no esperaba menos de ti, no parece tener muchas luces. Soy un fantasma, ¿no lo ves?, ¿o acaso no has visto nunca una peli de terror?.

3b. Ya perdona, es que una cosa es verlo por la tele y otra muy distinta verlo con mis propios ojos.

4b. Pues sí, soy un antiguo fantasma, que murió en estas tierras hace miles de años.

5b. ¡No me digas!. Y ahora has vuelto para maldecir a las personas que han profanado la tierra de tus antepasados construyendo este horrible lugar de color lila, no me sorprende. Es más, es lo que yo haría.

6b. Oh, no no no. Es cierto que esto está construido sobre un antiguo cementerio ibero donde enterrábamos a los nuestros, pero la verdad, de eso hace muchos años y el aburrimiento te hace olvidar muchas cosas, ¿sabes?.

7b. Ya claro, me imagino

8b. No, no te lo imaginas. En realidad, el que esto esté aquí me ha supuesto un poco de entretenimiento, aunque tampoco mucho, la verdad.

9b. Bueno, si hay algo que pueda hacer por ti para que te entretengas un rato.

10b. No, no te molestes, con esta conversación ya has hecho más amena mi aburrida eternidad. Aaaaah, como me gustaría poder ahogar mis penas con algo de alcohol.

11b. ¿Los fantasmas beben alcohol?.

12b. Desde que salió ese anuncio hace unos años de “Ron PimPom, la bebida que resucita a los muertos”, somos muchos los que nos hicimos adictos a la bebida.

13b. Vaya, y bueno, ¿funcionó?.

14b. No, pero al menos descubrimos una manera de ahogar las penas, o de pasar las penas, no sabes como pasa el tiempo sin que te enteres cuando tienes una cogerza.

15b. Que me vas a contar. En fin, si me sobra algo de bebida, te la traeré.

16b. ¿En serio?. Vaya gracias. Si puedo, buscaré alguna forma de recompensártelo.

17b. Bueno, no sabría para que me puede venir bien la ayuda de un fantasma, pero ya lo pensaré. En fin, tengo que irme.

18b. Adiós, y vuelve cuando quieras, no me viene mal una buena conversación de vez en cuando.

- Segunda vez que un PJ habla con él.

1. Lo siento amigo, estoy ocupado, me llaman por la otra línea, y es conferencia, mi tío desde el infierno.
2. Ah, bueno, pues ya vendré en otro momento.
3. Espera, ¿tienes algo de alcohol?.
4. Veré que puedo hacer con ese tema, ¿vale?.

5. De acuerdo, gracias... Si, si tío, si, te escucho... No, no, era un vivo, no le conoces...
- Conversación 2 (cuando le das una botella de cava).
 1. ¿Esto es para mí?. ¡Que bien, muchísimas gracias!.
 2. De nada, que la disfrutes, aunque no creo que te la agarres muy fuerte con eso.
 3. Da igual, menos da una piedra. Espera, déjame darte algo como agradecimiento.
 4. Bueno...
 5. Toma, este fémur, es de uno de mis antepasados, ¡ya no lo necesita!.
 6. Vaya gracias. ¿Seguro que me lo puedo quedar?.
 7. ¡Claro!. Los tengo a montones. Éramos muchos por aquel entonces, ¿sabes?.
 8. No, me refiero a si no me caerá una maldición por llevarme esto ni nada por el estilo, ¿no?.
 9. ¡Nah, eso son mariconadas!. Llévatelo, que no pasará nada.
 10. Vale, pues gracias.
 11. **Resultado:** ([“SACAR_A_INVENT”, “METER_A_INVENT”], [“Cava”, “Fémur”])

Ferretero

Historias en las que está involucrado: “El secreto del profesor”.

Parámetros:

- Nombre: “Ferretero”.
- Estados posibles: “Conversando, Esperando”

I. Acciones relevantes sobre este objeto (condición) : **Resultado** devuelto por el objeto.

- **Mirar ()**: (“¿Quién será este tipo?”. True, False, [], []).
- **Hablar (Estado = “Conversando”)**: (“Está ocupado hablando con otro, tendré que esperar...”, True, False, [], []).
- **Hablar (Estado = “Esperando”)**: Se inicia una conversación. Ver punto II.
- **Usar (Objeto Destino = Jabon con Estado = Molde)**: Se inicia una conversación. Ver punto II.
- **Fallo Acción ()**: (“No”, True, False, [], []).

II. Frases conversación:

- Primera vez y sucesivas que hablas con él. (Este PNJ ofrece siempre la misma conversación, su finalidad es cambiar el molde por copia de llave).
 1. Hola, ¿y tu quién eres?.
 2. Soy un ferretero, encantado de conocerte.
 3. ¿Un ferretero?, ¿y que pintas aquí?.
 4. No lo sé. Pregúntaselo a los programadores del juego.
 5. Me imaginaba una respuesta como esa.
 6. Bueno chico, cuando necesites ayuda con algún apaño de ferretería, ya sabes donde estoy, que me canso de hablar.
 7. Eh... bien, bueno es saberlo. Gracias.
 8. De nada, a mandar.
 9. ¡Hasta luego!.
- Cuando usas el molde con él:
 1. Ey amigo, esto tiene pinta de ser un trabajo para un buen ferretero.
 2. Esa era mi intención. ¿Puedes sacar una copia de esta llave?.
 3. Claro, tu sólo págame mis honorarios y tendrás lista tu copia.

4. De acuerdo, aquí tienes.
 5. Bien, así me gusta, espera un momento, no tardaré nada.
 6.
 7. Listo, aquí tienes, una copia de esa llave.
 8. Genial, gracias.
 9. De nada amigo, a mandar.
 10. ["SACAR_DE_INVY"] ["Jabon"], ["METER_A_INVY"], ["Llave"]
- Nota:** No se le quita dinero porque no se va a usar mucho más, pero queda constancia de que el jugador tiene un objeto "Dinero" que puede dar mucho de sí en posibles mejoras de la aplicación.

Arquitectura detallada

1. Arquitectura del sistema

PAQUETE: *agm.ClienteJugador*

ClienteJugador:

Es una clase que escucha en una cola Topic cuyo nombre es el id de la habitación en la que se encuentra el personaje jugador. Además cuando se solicita creación de personaje jugador o conexión de personaje jugador la recepción de mensajes en el cliente jugador se hace por la cola cuyo nombre es "topic/CONFIRMACIÓN".

Por esa cola puede recibir cuatro tipos de mensaje, notificaciones, ObjetosRefresco, ObjetoComunicación y Mensaje_Chat. En todos los casos, procesa esos mensajes, de la siguiente forma:

- Las notificaciones pueden ser pequeños cambios tales como añadir objetos que se han cogido al inventario, o quitar algunos que se pierden o usan, etc, todos ellos acompañados por mensajes textuales. Esta clase se encarga de mostrar esos mensajes pasándoselos al JTextArea *areaTexto* de la interfaz gráfica, y modifica el inventario del personaje jugador si es necesario, etc...
- Los objetosComunicacion se producen cuando se habla con los personajes no jugadores. Contienen dos LinkedList, uno de conversación fija y otro de opciones. Esta clase le pasa a *escritorTexto* la lista de conversación fija para que se muestre en la interfaz gráfica, y cuando se termina, si existen opciones, se muestran en la interfaz gráfica. Esta clase también se encarga de deshabilitar la opción de desconectar y ejecutar para que no puedas hacer nada más mientras estás hablando.
- Los objetos refresco incluyen toda la información necesaria para refrescar la interfaz gráfica ante un cambio de habitación por parte del personaje. Esta clase actualiza sus parámetros con la nueva información recibida, y llama al refresco de la interfaz gráfica, el cual usa los parámetros ya actualizados para realizar dicho refresco.
- Por último, cuando se recibe un Mensaje_Chat (de tipo message), simplemente se muestra el texto de su contenido en el JTextArea *mensajesEntrantes* que está dentro de un JScrollPane en la interfaz gráfica.

Por otro lado, esta clase cachea parte del estado de la habitación del intérprete cuando procesa peticiones de ejecutar acciones, y así consigue reducir el tráfico de red.

Atributos:

Estos son los atributos que guardan los estados del jugador:

- private PersonajeJugador pJ
- private java.util.List objetosEnPantalla

ClienteJugador crea y administra la interfaz gráfica en este parámetro:

- private static InterfazGrafica iG

Con los siguientes parámetros, se tiene información de la habitación donde está situado el jugador, y conexión con la cola de mensajes de esta:

- private HabitacionHome habitacionHome
- private Habitacion habitacion
- private InitialContext ctx
- private QueueConnection qC

- private TopicConnection tC

Los siguientes son parámetros necesarios para crear personajes y conectarse al mundo:

- private javax.jms.Queue colaKernel
- private TopicSubscriber subscriptor
- private boolean pedidoCrearPJ
- private boolean pedidaConexionPJ

Estos parámetros recogen las frases de acciones que realiza el jugador:

- private String objeto1IdFrase
- private String objeto2IdFrase
- private String idAccion
- private int estadoFrase -- Indica si la frase necesita dos objetos o solo uno.
- private String fondo – Imagen de la habitación.
- private Collection inventFijo – Objetos que siempre tienes.
- private String idPNJBloqueante – Si es null, no estás conversando con un personaje no jugador. En caso contrario, le es útil al clienteJugador para deshabilita la opción de desconectar y no dejar la conversación a medias.

Estos parámetros son usados para gestionar la interfaz gráfica:

- private int pixelsAltoPosicionables
- private int pixelsAnchoPosicionables
- private boolean respuestaRecibida
- private String textoConversacion
- private int posXRelativa
- private int posYRelativa
- private boolean mostrarBotonesOpciones
- private LinkedList opciones
- public static final String rutaImágenes

Métodos:

- private static String cargarIp() -- Carga la IP del servidor de un fichero XML.
- private static InitialContext getContext()
- private HabitacionHome getHabitacionHome() throws NamingException
- private void subscribeTopicHabitacion()

- `public InterfazGrafica creaInterfazGrafica(int resolucion)`
- `private void subscribeA(String topicString, String messageSelector)`
- `private void unsubscribeDeTopic(String topicString){`
- `private QueueConnection creaQueueConnection()`
- `private TopicConnection creaTopicConnection()`
- `public void solicitudCreacionPJ()`
- `public void solicitudConexionPJ()`
- `public void solicitudDesconexionPJ()`
- `public ClienteJugador`
- `public void onMessage(Message message)`
- `private void procesa(javax.jms.Message message)`
- `private PersonajeJugador buscarPersonaje(List lista,String idPJ)`
- `private void procesaObjeto(ObjetoActualizacion oAC)`
- `private void escribeConversacion(LinkedList conversacionFija,boolean empiezaJugador,String nombrePNJ)`
- `public void mensajeMostrado()`

Aquí vienen los métodos para procesar los mensajes recibidos por la cola de la habitación:

- `private void procesaObjeto(ObjetoComunicacion oC)`
- `private void procesaObjeto(ObjetosRefresco oR)`
- `private void procesaSacaPJ(SacaPJ sPJ)`
- `private void procesaCambioEstado(CambioEstado cE)`
- `private void interpretaMascara(String mascara)`
- `private void procesaMeterPJ(MeterPJ mPJ)`
- `private ObjetoEnPantalla buscarObjetoEnPantalla(String idO)`
- `private void procesaMeterInv(MeterInv ml)`
- `private void procesaSacarInv(SacarInv sl)`
- `private void procesaNotificacionTextual(NotificacionTextual nT)`
- `public java.util.List getObjetosEnPantalla()`
- `public String getObjeto1IdFrase()`
- `public String getObjeto2IdFrase()`

- `public void setObjeto1IdFrase(String pantalla)`
- `public void setObjeto2IdFrase(String pantalla)`
- `public String getFondo()`
- `public PersonajeJugador getPJ`
- `public int getEstadoFrase()`
- `public void setEstadoFrase(int i)`
- `public static void main(String []args)`
- `public List creaObjetosEnPantalla(java.util.List objetosNoPersonajes, java.util.List personajesNJugadores, java.util.List personajesJugadores)`

Los siguientes métodos tratan la ejecución de acciones por parte del jugador:

- `public void ejecutaAccion()` -- Este método es el que comprueba si la acción tiene sentido (no hablas con objetos inanimados, etc...) evitando así llamadas remotas innecesarias.
- `private boolean esPNJ(String idPNJ)`
- `private boolean esPJ(String idPJ)`
- `public void teRespondo(String idOpcion)` -- Similar a `ejecutarAccion`, pero limitado a conversaciones con personajes no jugadores, indica la opción elegida como contestación entre todas las posibles.
- `public void envioChat(String msg)`
- `public String getIdAccion()`
- `public void setIdAccion(String string)`
- `public void responder()`
- `public TopicSubscriber getSubscriber()`
- `public String getTextoConversacion()`
- `public void setSubscriber(TopicSubscriber subscriber)`
- `public void setTextoConversacion(String string)`
- `public String getIdPNJBloqueante()`

PAQUETE: agm.ClienteJugador.InterfazGrafica

InterfazGrafica:

Esta clase contiene la interfaz gráfica de la habitación.

La habitación, junto con sus objetos se dibuja en un panel situado en el centro de la ventana. En la parte superior se sitúan: la zona del inventario, en la que se muestran

todos los objetos que posee el jugador, y la zona de chat, donde el usuario puede escribir y leer los mensajes del chat de la habitación. En la parte inferior de la ventana se encuentran los botones necesarios para poder ejecutar acciones.

Las únicas llamadas que hace la interfazGrafica sobre clienteJugador son tres: *ejecutarAcción*, para informar de las acciones que quiere llevar a cabo el usuario, *TeRespondo*, para responder en conversaciones fijas, y *EnvioChat*, para hablar con otros personajes jugadores. Con ellas, esta clase recoge todas las interacciones posibles del jugador con el mundo virtual de AGM.

Atributos:

- public final static int r1024_768
- public final static int r800_600
- private ClienteJugador cJ
- private ConfigIG configIG
- private static Bot pantalla
- private static JTextArea areaTexto
- private static EscritorTexto escritorTexto
- private static JSplitPane separaPantallaJuegoYConsolaNoEditable
- private JSplitPane separaInventYConsolaEditable
- private JTextArea mensajesEntrantes
- private JTextField campoTexto
- private JButton botonEnviar
- private Hashtable posicionAObjeto
- private JTextArea accionFrase
- private JTextArea objeto1
- private JTextArea con
- private JTextArea objeto2
- private boolean fraseTemporal
- private JSplitPane separaConsolaEditableYPantalla
- private JTextArea objetoSeleccionado
- private JButton botonEjecutar
- private Box cajaOpciones
- private JButton botonDesconectar
- private JTextArea campoDeTexto

- private JButton []botonesOpciones
- private OyenteOpcionMouse []oyenteOpcion
- private Box hB1
- private JSplitPane jSplitP

Métodos:

- private void cambiaFrase(ObjetoEnPantalla objeto)
- private InitialContext getContext()
- public void refrescarInventario()
- private JScrollPane getInventario(boolean generarInventFijo)
- private Box crearInventario()
- public void meteAInv(ObjetoEnPantalla o)
- public void sacaDeInv(String idO)
- private Box creaCajaChat()
- public Box crearPantallaJuego()
- private Box crearConsolaNoEditable
- private Box creaCajaPantallaJuegoYConsolaNoEditable(Box pantallaJuego, Box consolaNoEditable)
- private Box crearCajaFrase(Color colorFondo,Font f)
- private JButton creaBotonAccion(String accion,Font fuente,boolean asociarOyenteAccion,String formato)
- public Box crearCajaAcciones(Font f)
- public Box crearCajaFraseYAccionesYOpciones(Color c, Font f)
- private JSplitPane configuraInterfazGrafica()
- public InterfazGrafica(ClienteJugador cJ, int resolucion)
- public void escribirTexto(String texto)
- public void escribirTexto(String texto,Color c, boolean esLaUltimaLinea)
- private void ocultaConsolaEditable
- private void muestraConsolaEditable()
- public void eliminarInterfazGrafica()
- public void mostrarPantallaInicio
- public void escribirMensajeDeChat(String nombrePJ,String mensaje)

- `private int tiempo(int length)`
- `public Box crearCajaOpciones()`
- `public void mostrarBotonesOpciones(LinkedList opciones)`
- `public void ocultarBotonesOpciones`
- `public void setEnableBotonEjecutar(boolean enable`
- `public void setEnableBotonEnvioChat(boolean enable)`
- `public void setEnableBotonDesconectar(boolean enable)`
- `public void repaint()`

PAQUETE: agm.ClienteAdministrador

ClienteAdministrador:

Es una clase que se encarga de inicializar el mundo del juego, sirviéndose para ello del bean GestorSistema.

Atributos:

- `private static GestorSistema gestorSistema`

Métodos:

- `public ClienteAdministrador()`
- `private static InitialContext getContext()`
- `private GestorSistemaHome getGestorSistemaHome()`
- `private void borraMundo()`
- `private void creaMundo()`
- `public static void main(String []args)`

PAQUETE: agm.objetos

ObjetoComunicacion:

Es la clase que se utiliza para enviar al cliente jugador los datos de una conversación con un personaje no jugador.

Atributos:

- `private LinkedList conversacionFija`

- private LinkedList opciones
- private boolean empiezaJugador
- private String idPJ
- private String nombrePNJ

Métodos:

- public void setEmpiezaJugador(boolean e)
- public void setConversacionFija(LinkedList l)
- public void setOpciones(LinkedList l)
- public String getIdPJ()
- public void setIdPJ(String string)
- public LinkedList getConversacionFija()
- public boolean isEmpiezaJugador()
- public LinkedList getOpciones()
- public String getNombrePNJ()
- public void setNombrePNJ(String string)

ObjetoRefresco:

Es la clase que se utiliza para enviar al cliente jugador la información necesaria cuando su personaje cambia de habitación.

Atributos:

- private java.util.List objetosNoPersonaje;
- private java.util.List personajesNJugadores;
- private java.util.List personajesJugadores;
- private String fondo;
- private String idHab;
- private String idPJ;

Métodos:

- public String getFondo()
- public void setFondo(String string)
- public java.util.List getPersonajesJugadores()

- `public java.util.List getPersonajesNJugadores()`
- `public void setPersonajesJugadores(java.util.List list)`
- `public void setPersonajesNJugadores(java.util.List list)`
- `public java.util.List getObjetosNoPersonaje()`
- `public void setObjetosNoPersonaje(java.util.List list)`
- `public String getIdHab`
- `public void setIdHab(String string)`
- `public String getIdPJ()`
- `public void setIdPJ(String string)`

NotificacionTextual:

Es la clase que se utiliza para enviar notificaciones al cliente jugador.

Atributos:

- `protected String textoAMostrar`
- `protected String idPJ`

Métodos:

- `public NotificacionTextual(String textoAMostrar, String idPJ)`
- `public String getIdPJ()`
- `public String getTextoAMostrar`
- `public void setIdPJ(String string)`
- `public void setTextoAMostrar(String string)`

MeterInv:

Clase que hereda de `NotificacionTextual` y que sirve para enviar al cliente jugador la información necesaria cuando se añade un objeto a su inventario.

Atributos:

- `private ObjetoNoPersonajeClase o`

Métodos:

- `public MeterInv(String textoAMostrar, String idPJ, ObjetoNoPersonajeClase o)`
- `public ObjetoNoPersonajeClase getO()`

SacarInv:

Clase que hereda de `NotificacionTextual` y que sirve para enviar al cliente jugador la información necesaria cuando se saca un objeto de su inventario.

Atributos:

- `private ObjetoNoPersonajeClase o`

Métodos:

- `public SacarInv(String textoAMostrar, String idPJ, ObjetoNoPersonajeClase o)`
- `public SacarInv(String textoAMostrar, String idPJ)`
- `public ObjetoNoPersonajeClase getO()`

MetePJ:

Clase que hereda de `NotificacionTextual` y que sirve para enviar a los clientes jugadores que están en una habitación la información necesaria cuando un nuevo personaje entra en ella.

Atributos:

- `private PersonajeJugador pj`
- `private String textoAMostrarAlResto`
- `private String nombre`
- `private String imagen`
- `private String formato`

Métodos:

- `public MetePJ(String textoAMostrar, String textoAMostrarAlResto, String nombre, String idPJ, String imagen, String formato)`
- `public String getNombre()`
- `public void setNombre(String string)`
- `public String getIdPJ()`
- `public void setIdPJ(String string)`
- `public String getFormato()`
- `public String getImagen()`
- `public PersonajeJugador getPJ()`
- `public String getTextoAMostrarAlResto()`

- `public void setFormato(String string)`
- `public void setImagen(String string)`
- `public void setPJ(PersonajeJugador jugador)`
- `public void setTextoAMostrarAlResto(String string)`

SacaPJ:

Esta clase hereda de `NotificaciónTextual` y se utiliza para enviar a los clientes jugadores que están en una habitación la información necesaria cuando un personaje sale de ella.

Atributos:

- `protected String textoAMostrarAlResto`
- `protected String nombre`

Métodos:

- `public SacaPJ(String textoAMostrar, String textoAMostrarAlResto, String idPJ, String nombre)`
- `public String getTextoAMostrarAlResto()`
- `public void setTextoAMostrarAlResto(String string)`

CambioEstado:

Esta clase hereda de `NotificaciónTextual` y se utiliza para enviar al cliente la información necesaria cuando un objeto cambia de estado.

Atributos:

- `protected String idO`
- `protected String estado`

Métodos:

- `public CambioEstado(String textoAMostrar, String idPJ)`
- `public CambioEstado(String textoAMostrar, String idPJ, String idO, String estado)`
- `public String getIdO()`
- `public void setIdO(String string)`
- `public String getEstado()`
- `public void setEstado(String string)`

ObjetoActualizacion:

Esta clase se utiliza para almacenar varias notificaciones generadas al ejecutar una acción, antes de enviarle al cliente la información.

Atributos:

- private LinkedList NotificacionesTextuales

Métodos:

- public LinkedList getNotificacionesTextuales()
- public void setNotificacionesTextuales(LinkedList list)

agm.objetos.objetosNoPersonaje

ObjetoNoPersonajeClase:

Esta clase representa un objeto no personaje del juego.

Atributos:

- private String idO
- protected String estado
- protected String idHab
- private int posX
- private int posY
- protected String tipo
- private String nombre

Métodos:

- public ObjetoNoPersonajeClase()
- public ObjetoNoPersonajeClase(String idO,String estado,String idHab,int posX,int posY)
- public String getTipo()
- public abstract Acciones ejecutarAccion(String accion, String instrumento, String idPJ)
- public String getEstado()
- public String getIdHab()
- public String getIdO()

- public int getPosX()
- public int getPosY()
- public void setEstado(String string)
- public void setIdHab(String string)
- public void setIdO(String string)
- public void setPosX(int i)
- public void setPosY(int i)
- public abstract void cambiaEstado()
- public String getNombre()
- public void setNombre(String string)
- public boolean equals(Object o)

Jabon:

Esta clase hereda de ObjetoNoPersonajeClase y representa un objeto jabón.

Métodos:

- public Jabon()
- public Jabon(String idO,String estado,String idHab,int posX,int posY)
- public void cambiaEstado()
- public Acciones ejecutarAccion(String idA, String IdOd, String IdPj)

Llave:

Esta clase hereda de ObjetoNoPersonajeClase y representa un objeto llave.

Métodos:

- public Llave ()
- public Llave(String idO,String estado,String idHab,int posX,int posY)
- public void cambiaEstado()
- public Acciones ejecutarAccion(String idA, String IdOd, String IdPj)

Aprobado:

Esta clase hereda de ObjetoNoPersonajeClase y representa un objeto aprobado.

Métodos:

- public Aprobado ()
- public Aprobado (String idO,String estado,String idHab,int posX,int posY)
- public void cambiaEstado()
- public Acciones ejecutarAccion(String idA, String IdOd, String IdPj)

Pastillero:

Esta clase hereda de ObjetoNoPersonajeClase y representa un objeto pastillero de jabón.

Atributos:

- private LinkedList PJs

Métodos:

- public Pastillero ()
- public Pastillero (String idO,String estado,String idHab,int posX,int posY)
- public void cambiaEstado()
- public Acciones ejecutarAccion(String idA, String IdOd, String IdPj)
- public void cargarLista()
- public void guardarLista()
- public void setPJs(LinkedList list)
- public void setIdO()

Puerta:

Esta clase hereda de ObjetoNoPersonajeClase y representa un objeto puerta con el comportamiento de una puerta normal, es decir, cambiar de habitación.

Atributos:

- protected String idHabDest

Métodos:

- public Puerta ()
- public Puerta (String idO,String estado,String idHab,int posX,int posY)
- public void cambiaEstado()
- public Acciones ejecutarAccion(String idA, String IdOd, String IdPi)
- public void setIdHabDest(String string)
- public String getIdHabDest()

PuertaADespacho:

Esta clase hereda de Puerta y representa a la puerta que da acceso al despacho.

Métodos:

- public PuertaADespacho ()
- public Puerta ADespacho(String idO,String estado,String idHab,int posX,int posY)
- public void cambiaEstado()
- public Acciones ejecutarAccion(String idA, String IdOd, String IdPi)

PuertaAConserjeria:

Esta clase hereda de Puerta y representa la puerta que da acceso a conserjería.

Atributos:

- protected String idConserje

Métodos:

- public Puerta ()
- public Puerta (String idO,String estado,String idHab,int posX,int posY)
- public void cambiaEstado()
- public Acciones ejecutarAccion(String idA, String IdOd, String IdPi)
- public void guardarConserje()
- public void cargarConserje()
- public void setIdConserje(String string)

- `public void setIdO()`
- `public String getIdConserje()`

Acciones:

Esta clase sirve para encapsular las acciones que debe realizar una habitacion, después de haber ejecutado una acción sobre un objeto contenido en ella.

Atributos:

- `protected LinkedList parametros`

Métodos:

- `public abstract void ejecutar(String idHab,String idPJ)`
- `public abstract boolean interpretaCondicion(Habitacion hab1,Habitacion hab2,String idPJ)`
- `public String meterEnInventario(Habitacion hab,String idPJ,String tipoObjeto,String textoAMostrar)`
- `public void sacarDelInventario(Habitacion hab,String idO,String idPJ,String textoAMostrar)`
- `public void cambiarEstadoObjeto(Habitacion hab,String idO,String idPJ, String textoAMostrar)`
- `public void cambioHabitacion(Habitacion hab1,Habitacion hab2,String idPJ)`
- `public boolean cambioHabitacionCondicional(Habitacion hab1,Habitacion hab2,String idPJ)`
- `public void mostrarResultado(Habitacion hab,agm.objetos.ObjetoActualizacion oA)`
- `public void mostrarResultado(Habitacion hab,agm.objetos.ObjetoComunicacion oC)`
- `public void mostrarResultado(Habitacion hab,agm.objetos.ObjetosRefresco oR)`
- `public LinkedList getParametros`
- `public void setParametros(LinkedList list)`
- `private static InitialContext getContext() throws NamingException`
- `public static HabitacionHome getHabitacionHome()`
- `public static Habitacion conseguirHabitacion(String idHab)`
- `public ObjetoActualizacion generaNotificacion(String idPJ,String texto)`
- `public ObjetosRefresco generaRefresco(Habitacion h, String idPJ)`

agm.objetos.personajesNoJugadores

PersonajeNoJugadorClase:

Esta clase representa un personaje no jugador del juego.

Atributos:

- private String idPNJ
- protected String estado
- protected String idHab
- private int posX
- private int posY
- protected String tipo
- private String nombre

Métodos:

- public PersonajeNoJugadorClase ()
- public PersonajeNoJugadorClase (String idPNJ,String nombre,String idHab,int posX,int posY,String estado)
- public String getTipo()
- public abstract Acciones String idA, String IdO, String IdPj)
- public abstract Acciones teRespondo(String opcion, String idPJ)
- public String getEstado()
- public String getIdHab()
- public String getIdPNJ()
- public int getPosX()
- public int getPosY()
- public String getIdHab()
- public void setEstado(String string)
- public void setIdHab(String string)
- public void setIdO(String string)
- public void setPosX(int i)
- public void setPosY(int i)

- public abstract void cambiaEstado()
- public String getNombre()
- public void setNombre(String string)

Ferretero:

Esta clase hereda de PersonajeNoJugadorClase y representa a un ferretero.

Métodos:

- public Ferretero ()
- public Ferretero (String idPNJ,String nombre,String idHab,int posX,int posY,String estado)
- public void cambiaEstado()
- public abstract Acciones ejecutarAccion(String idA, String IdO, String IdPj)
- public abstract Acciones teRespondo(String opcion, String idPJ)

Conserje:

Esta clase hereda de PersonajeNoJugadorClase y representa a un conserje de la facultad.

Métodos:

- public Conserje()
- public Conserje (String idPNJ,String nombre,String idHab,int posX,int posY,String estado)
- public void cambiaEstado()
- public abstract Acciones ejecutarAccion(String idA, String IdO, String IdPj)
- public abstract Acciones teRespondo(String opcion, String idPJ)

ProfesorX:

Esta clase hereda de PersonajeNoJugadorClase y representa a un profesor de la facultad.

Atributos:

- private LinkedList PJs

Métodos:

- public ProfesorX ()
- public ProfesorX (String idPNJ,String nombre,String idHab,int posX,int posY,String estado)
- public void cambiaEstado()
- public abstract Acciones ejecutarAccion(String idA, String IdO, String IdPj)
- public abstract Acciones teRespondo(String opcion, String idPJ)
- public void guardarLista()
- public void cargarLista()

Frase:

Esta clase se utiliza para almacenar una frase junto con su identificador, y así mandarsela al cliente para que elija la opción deseada.

Atributos:

- private String idFrase
- private String contenido

Métodos:

- public String getContenido
- public String getIdFrase()
- public void setContenido(String string)
- public void setIdFrase(String string) public Frase(String contenido,String idF)

agm.objetos.personajesJugadores

PersonajeJugador:

Esta clase representa a un jugador participante del juego.

Atributos:

- private String idP
- private String nombre
- private String password
- private float dinero

- private String idHab
- private int posX
- private int posY
- private Collection objetos
- private String imagen
- private String formatImagen

Métodos:

- public PersonajeJugador(String idP,String nombre, String password, String imagen, float dinero, String idHab, int posX,int posY,Collection objetos)
- public String getIdHab()
- public String getIdP()
- public String getNombre()
- public Collection getObjetos()
- public String getPassword()
- public int getPosX()
- public int getPosY()
- public void setDinero(float f)
- public void setIdP(String string)
- public void setNombre(String string)
- public void setObjetos(Collection collection)
- public void setPassword(String string)
- public void setPosX(int i)
- public void setPosY(int i)
- public String getImagen()
- public void setImagen(String string)
- public String getFormatImagen()
- public void setFormatImagen(String string)

PAQUETE: *agm.servidor*

agm.servidor.habitaciones

HabitacionBean:

Es un bean de entidad que representa a las habitaciones del juego con todo el contenido que hay en ellas. La clase es abstracta.

Se encarga de responder a las peticiones de ejecutar acciones sobre los objetos que se encuentran en ella llamando al método correspondiente del objeto en cuestión. De este recibe una clase anónima *acciones* que le suministra el resultado de la acción que se ha ejecutado, el cual procesa, creando notificaciones si es preciso y mandándolas a su cola para que las procesen los *clienteJugador* a los que vaya dirigido.

En dichos casos, cuando tiene que mandar alguna información a los clientes jugadores cuyos personajes se encuentran en dicha habitación, envía un mensaje por el Topic cuyo nombre es el identificador de la habitación.

Atributos:

Estos son atributos propios de la habitación, y definen sus características:

- private String idHab
- private String nombre
- private boolean luz
- private int ancho
- private int alto
- private String mascara -- Define con cuatro valores una matriz de posiciones expresadas en pixeles por las que se pueden mover los jugadores en dicha habitación.
- private Collection puertas – Lista de objetos puerta, que en el diseño de objetos se mencionan como puertaGenerica, y que se limitan a comunicar las habitaciones del mundo.
- private java.util.List objetosNoPersonaje -- Lista de objetos de la habitación.
- private Collection personajesNJ -- Lista de personajes no jugadores de la habitación.
- private Collection personajesJ -- Lista de personajes jugadores que se encuentran en la habitación.

Atributos para conectar con la cola en la que escuchan todos los clientes que están en esa habitación:

- private TopicPublisher publisher
- private TopicConnection tC
- private TopicSession tS

Métodos:

- `private void invocaServicioMBean(String servicio, String nombreMetodo, String[] tiposParametros, Object[] valorParametros)`
- `public void ejbRemove(HabitacionPK pk)`
- `public HabitacionPK ejbCreate(String idHab, String nombre, boolean luz,int ancho,int alto,String mascara,Collection puertas)`
- `public void ejbLoad()`
- `private TopicSession creaTopicSession()`
- `public int getAlto()`
- `public int getAncho()`
- `public String getIdHab()`
- `public boolean isLuz()`
- `public String getMascara()`
- `public String getNombre()`
- `public Collection getPuertas()`
- `public void setAlto(int i)`
- `public void setAncho(int i)`
- `public void setIdHab(String string)`
- `public void setLuz(boolean b)`
- `public void setMascara(String string)`
- `public void setNombre(String string)`
- `public void setPuertas(Collection collection)`
- `public HabitacionData getHabitacionData()`
- `public HabitacionPK ejbFindByPrimaryKey(HabitacionPK pk)`
- `public HabitacionPK ejbFindByIdPJ(String pk)`
- `public HabitacionPK ejbFindByIdPNJ(String idP)`
- `public HabitacionPK ejbFindByIdO(String idO)`
- `public Collection ejbFindAll()`
- `public Collection ejbFindByNombre(String nombre)`
- `public void ejecutaAccion(String idAccion,String idActor,String idO1, String idO2)`

- `public void teRespondo(String idOpcion, String idPJ, String idPNJBloqueante`
- `private QueueConnection creaQueueConnection()`
- `public void envioChat(String msg, String nombrePJOrador)`
- `public java.util.List getObjetosNoPersonaje()`
- `public void setObjetosNoPersonaje(java.util.List collection)`
- `public Collection getPersonajesJ()`
- `public Collection getPersonajesNJ()`
- `public void setPersonajesJ(Collection collection)`
- `public void setPersonajesNJ(Collection collection)`
- `public void metePersonajeJ(PersonajeJugador pJ,String textoAMostrarAPJ,String textoAMostrarAlResto)`
- `public PersonajeJugador sacaPersonajeJ(String idPJ, String textoAMostrar,String textoAMostrarAlResto)`
- `private PersonajeJugador buscarPersonaje(String idPJ)`
- `public void meterEnInventario(String idPJ,ObjetoNoPersonajeClase objeto,String textoAMostrar)`
- `public void sacarDelInventario(String idO,String idPJ,String textoAMostrar)`
- `public void cambiarEstadoObjeto(String idO,String idPJ,String textoAMostrar)`
- `public ObjetoNoPersonajeClase buscarObjeto(String idO, String idPJ)`
- `public PersonajeNoJugadorClase buscarPNJEnHabitacion(String idO)`
- `private ObjetoNoPersonajeClase buscarObjetoEnHabitacion(String idO)`
- `private ObjetoNoPersonajeClase buscarObjetoEnInventario(String idO,String idPJ)`
- `public void mostrarResultado(agm.objetos.ObjetoActualizacion oA)`
- `public void mostrarResultado(agm.objetos.ObjetosRefresco oR)`
- `public void mostrarResultado(agm.objetos.ObjetoComunicacion oC)`
- `public void meteObjetoNP(ObjetoNoPersonajeClase oNP)`
- `public void sacaObjetoNP(ObjetoNoPersonajeClase oNP)`
- `public String ejbHomeGetClaveLibreParaObjetoNP()`
- `public void setPublisher(TopicPublisher publisher)`
- `public void setSession(TopicSession session)`

Habitacion:

Es la interfaz remota del bean Habitacion.

Métodos:

- `public void ejbRemove(agm.servidor.habitaciones.HabitacionPK pk)`
- `public void ejbLoad()`
- `public int getAlto()`
- `public int getAncho()`
- `public java.lang.String getIdHab()`
- `public boolean isLuz()`
- `public java.lang.String getMascara()`
- `public java.lang.String getNombre()`
- `public java.util.Collection getPuertas()`
- `public void setAlto(int i)`
- `public void setAncho(int i)`
- `public void setIdHab(java.lang.String string)`
- `public void setLuz(boolean b)`
- `public void setMascara(java.lang.String string)`
- `public void setNombre(java.lang.String string)`
- `public void setPuertas(java.util.Collection collection)`
- `public void ejecutaAccion(java.lang.String idAccion,java.lang.String idActor,java.lang.String idO1,java.lang.String idO2)`
- `public void teRespondo(java.lang.String idOpcion,java.lang.String idPJ,java.lang.String idPNJBloqueante)`
- `public void envioChat(java.lang.String msg,java.lang.String nombrePJOrador)`
- `public java.util.List getObjetosNoPersonaje()`
- `public void setObjetosNoPersonaje(java.util.List collection)`
- `public java.util.Collection getPersonajesJ()`
- `public java.util.Collection getPersonajesNJ()`
- `public void setPersonajesJ(java.util.Collection collection)`
- `public void setPersonajesNJ(java.util.Collection collection)`

- `public void metePersonajeJ(agm.objetos.personajesJugadores.PersonajeJugador pJ,java.lang.String textoAMostrarAPJ,java.lang.String textoAMostrarAlResto)`
- `public agm.objetos.personajesJugadores.PersonajeJugador sacaPersonajeJ(java.lang.String idPJ,java.lang.String textoAMostrar,java.lang.String textoAMostrarAlResto)`
- `public void meterEnInventario(java.lang.String idPJ,agm.objetos.objetosNoPersonaje.ObjetoNoPersonajeClase objeto,java.lang.String textoAMostrar)`
- `public void sacarDelInventario(java.lang.String idO,java.lang.String idPJ,java.lang.String textoAMostrar)`
- `public void cambiarEstadoObjeto(java.lang.String idO,java.lang.String idPJ,java.lang.String textoAMostrar)`
- `public agm.objetos.objetosNoPersonaje.ObjetoNoPersonajeClase buscarObjeto(java.lang.String idO,java.lang.String idPJ)`
- `public agm.objetos.personajesNoJugadores.PersonajeNoJugadorClase buscarPNJEnHabitacion(java.lang.String idO)`
- `public void mostrarResultado(agm.objetos.ObjetoActualizacion oA)`
- `public void mostrarResultado(agm.objetos.ObjetosRefresco oR)`
- `public void mostrarResultado(agm.objetos.ObjetoComunicacion oC)`
- `public void meteObjetoNP(agm.objetos.objetosNoPersonaje.ObjetoNoPersonajeClase oNP)`
- `public void sacaObjetoNP(agm.objetos.objetosNoPersonaje.ObjetoNoPersonajeClase oNP)`

HabitacionBMP:

Es una clase que hereda de Habitacion bean. Es un BMP, es decir la persistencia está gestionada por el bean. Es generado por xdoclet a partir de HabitacionBean.

Atributos:

- `private boolean dirty`
- `private javax.ejb.EntityContext ctx`
- `private static agm.servidor.habitaciones.HabitacionDAO dao`

Métodos:

- `public int getAlto()`
- `public int getAncho()`

- `public String getIdHab()`
- `public boolean isLuz()`
- `public String getMascara()`
- `public String getNombre()`
- `public Collection getPuertas()`
- `public void setAlto(int i)`
- `public void setAncho(int i)`
- `public void setIdHab(String string)`
- `public void setLuz(boolean b)`
- `public void setMascara(String string)`
- `public void setNombre(String string)`
- `public void setPuertas(Collection collection)`
- `public java.util.List getObjetosNoPersonaje()`
- `public void setObjetosNoPersonaje(java.util.List collection)`
- `public Collection getPersonajesJ()`
- `public Collection getPersonajesNJ()`
- `public void setPersonajesJ(Collection collection)`
- `public void setPersonajesNJ(Collection collection)`
- `public boolean isModified()`
- `protected void makeDirty()`
- `protected void makeClean()`
- `public agm.servidor.habitaciones.HabitacionData getData()`
- `public agm.servidor.habitaciones.HabitacionPK ejbCreate(java.lang.String idHab,java.lang.String nombre,boolean luz,int ancho,int alto,java.lang.String mascara,java.util.Collection puertas)`
- `public void ejbPostCreate(java.lang.String idHab,java.lang.String nombre,boolean luz,int ancho,int alto,java.lang.String mascara,java.util.Collection puertas)`
- `public agm.servidor.habitaciones.HabitacionPK ejbFindByPrimaryKey(agm.servidor.habitaciones.HabitacionPK pk)`
- `public agm.servidor.habitaciones.HabitacionPK ejbFindByIdPJ(java.lang.String pk)`
- `public agm.servidor.habitaciones.HabitacionPK ejbFindByIdPNJ(java.lang.String idP)`
- `public agm.servidor.habitaciones.HabitacionPK ejbFindByIdO(java.lang.String idO)`

- `public java.util.Collection ejbFindAll()`
- `public java.util.Collection ejbFindByNombre(java.lang.String nombre)`
- `public void ejbLoad()`
- `public void ejbStore()`
- `public void ejbActivate()`
- `public void ejbPassivate()`
- `public void setEntityContext(javax.ejb.EntityContext ctx)`
- `public void unsetEntityContext()`
- `public void ejbRemove()`
- `protected static synchronized agm.servidor.habitaciones.HabitacionDAO getDao()`
- `public java.lang.String ejbHomeGetClaveLibreParaObjetoNP()`

HabitacionDAO:

Es una interfaz que contiene los *Métodos* de acceso a la base de datos que usa el bean *Habitacion*. Es generada por Xdoclet a partir de *HabitacionBean*.

Métodos:

- `public void init()`
- `public void load (agm.servidor.habitaciones.HabitacionPK pk, agm.servidor.habitaciones.HabitacionBean ejb)`
- `public void store(agm.servidor.habitaciones.HabitacionBean ejb)`
- `public void remove(agm.servidor.habitaciones.HabitacionPK pk)`
- `public agm.servidor.habitaciones.HabitacionPK create (agm.servidor.habitaciones.HabitacionBean ejb)`
- `public agm.servidor.habitaciones.HabitacionPK findByPrimaryKey (agm.servidor.habitaciones.HabitacionPK pk)`
- `public agm.servidor.habitaciones.HabitacionPK findByIdPJ(java.lang.String pk)`
- `public agm.servidor.habitaciones.HabitacionPK findByIdPNJ(java.lang.String idP)`
- `public agm.servidor.habitaciones.HabitacionPK findByIdO (java.lang.String idO)`
- `public java.util.Collection findAll()`
- `public java.util.Collection findByNombre(java.lang.String nombre)`
- `public java.lang.String getClaveLibreParaObjetoNP()`

HabitacionHome:

Es la interfaz Home del bean Habitacion.

Atributos:

- public static final String COMP_NAME
- public static final String JNDI_NAME

Métodos:

- public agm.servidor.habitaciones.Habitacion create(java.lang.String idHab , java.lang.String nombre , boolean luz , int ancho , int alto , java.lang.String mascara , java.util.Collection puertas)
- public agm.servidor.habitaciones.Habitacion findByPrimaryKey (agm.servidor.habitaciones.HabitacionPK pk)
- public agm.servidor.habitaciones.Habitacion findByIdPJ(java.lang.String pk)
- public agm.servidor.habitaciones.Habitacion findByIdPNJ(java.lang.String idP)
- public agm.servidor.habitaciones.Habitacion findByIdO(java.lang.String idO)
- public java.util.Collection findAll()
- public java.util.Collection findByNombre(java.lang.String nombre)
- public java.lang.String getClaveLibreParaObjetoNP()

HabitacionLocal:

Es la interfaz local del bean Habitacion.

Métodos:

- public void ejbRemove(agm.servidor.habitaciones.HabitacionPK pk)
- public void ejbLoad()
- public int getAlto()
- public int getAncho()
- public java.lang.String getIdHab()
- public boolean isLuz()
- public java.lang.String getMascara()
- public java.lang.String getNombre()

- `public java.util.Collection getPuertas()`
- `public void setAlto(int i)`
- `public void setAncho(int i)`
- `public void setIdHab(java.lang.String string)`
- `public void setLuz(boolean b)`
- `public void setMascara(java.lang.String string)`
- `public void setNombre(java.lang.String string)`
- `public void setPuertas(java.util.Collection collection)`
- `public void ejecutaAccion(java.lang.String idAccion,java.lang.String idActor,java.lang.String idO1,java.lang.String idO2)`
- `public void teRespondo(java.lang.String idOpcion,java.lang.String idPJ,java.lang.String idPNJBloqueante)`
- `public void envioChat(java.lang.String msg,java.lang.String nombrePJOrador)`
- `public java.util.List getObjetosNoPersonaje()`
- `public void setObjetosNoPersonaje(java.util.List collection)`
- `public java.util.Collection getPersonajesJ()`
- `public java.util.Collection getPersonajesNJ()`
- `public void setPersonajesJ(java.util.Collection collection)`
- `public void setPersonajesNJ(java.util.Collection collection)`
- `public void metePersonajeJ(agm.objetos.personajesJugadores.PersonajeJugador pJ,java.lang.String textoAMostrarAPJ,java.lang.String textoAMostrarAlResto)`
- `public agm.objetos.personajesJugadores.PersonajeJugador sacaPersonajeJ(java.lang.String idPJ,java.lang.String textoAMostrar,java.lang.String textoAMostrarAlResto)`
- `public void meterEnInventario(java.lang.String idPJ,agm.objetos.objetosNoPersonaje.ObjetoNoPersonajeClase objeto,java.lang.String textoAMostrar)`
- `public void sacarDelInventario(java.lang.String idO,java.lang.String idPJ,java.lang.String textoAMostrar)`
- `public void cambiarEstadoObjeto(java.lang.String idO,java.lang.String idPJ,java.lang.String textoAMostrar)`
- `public agm.objetos.objetosNoPersonaje.ObjetoNoPersonajeClase buscarObjeto(java.lang.String idO,java.lang.String idPJ)`
- `public agm.objetos.personajesNoJugadores.PersonajeNoJugadorClase buscarPNJEnHabitacion(java.lang.String idO)`
- `public void mostrarResultado(agm.objetos.ObjetoActualizacion oA)`

- `public void mostrarResultado(agm.objetos.ObjetosRefresco oR)`
- `public void mostrarResultado(agm.objetos.ObjetoComunicacion oC)`
- `public void meteObjetoNP(agm.objetos.objetosNoPersonaje.ObjetoNoPersonajeClase oNP)`
- `public void sacaObjetoNP(agm.objetos.objetosNoPersonaje.ObjetoNoPersonajeClase oNP)`

HabitacionLocalHome:

Es la interfaz Home local del bean Habitacion.

Atributos:

- `public static final String COMP_NAME`
- `public static final String JNDI_NAME`

Métodos:

- `public agm.servidor.habitaciones.Habitacion create(java.lang.String idHab , java.lang.String nombre , boolean luz , int ancho , int alto , java.lang.String mascara , java.util.Collection puertas)`
- `public agm.servidor.habitaciones.Habitacion findByPrimaryKey (agm.servidor.habitaciones.HabitacionPK pk)`
- `public agm.servidor.habitaciones.Habitacion findByIdPJ(java.lang.String pk)`
- `public agm.servidor.habitaciones.Habitacion findByIdPNJ(java.lang.String idP)`
- `public agm.servidor.habitaciones.Habitacion findByIdO(java.lang.String idO)`
- `public java.util.Collection findAll()`
- `public java.util.Collection findByNombre(java.lang.String nombre)`
- `public java.lang.String getClaveLibreParaObjetoNP()`

HabitacionPK:

Es la clase que representa al primary key (o clave primaria) del bean Habitacion.

Atributos:

- `private int _hashCode`
- `private StringBuffer _toStringValue`
- `public java.lang.String idHab`

Métodos:

- public HabitaciónPK()
- public HabitaciónPK(java.lang.String idHab)
- public java.lang.String getIdHab()
- public void setIdHab(java.lang.String idHab)
- public int hashCode()
- public boolean equals(Object obj)
- public String toString()

HabitacionUtil:

Es un una clase de utilidad con métodos estáticos que sirve principalmente para obtener las interfaces Home remota y local del bean Habitación.

Atributos:

- private static agm.servidor.habitaciones.HabitacionHome cachedRemoteHome
- private static agm.servidor.habitaciones.HabitacionLocalHome cachedLocalHome
- private static String hexServerIP
- private static final SecureRandom seeder

Métodos:

- public static agm.servidor.habitaciones.HabitacionHome getHome()
- public static agm.servidor.habitaciones.HabitacionHome getHome(Hashtable environment)
- public static agm.servidor.habitaciones.HabitacionLocalHome getLocalHome()
- public static final String generateGUID(Object o)
- private static int getInt(byte bytes[])
- private static String hexFormat(int i, int j)

HabitacionData:

Es un una clase que encapsula todos los *Atributos* del bean Habitación.

Atributos:

- private String idHab

- private String nombre
- private boolean luz
- private int ancho
- private int alto
- private String mascara
- private Collection puertas
- private java.util.List objetosNoPersonaje
- private Collection personajesNJ
- private Collection personajesJ

Métodos:

- public HabitaciónData()
- public HabitaciónData(int alto,int ancho,java.lang.String idHab,boolean luz,java.lang.String mascara,java.lang.String nombre,java.util.Collection puertas,java.util.List objetosNoPersonaje,java.util.Collection personajesJ,java.util.Collection personajesNJ)
- public HabitaciónData(HabitaciónData otherData)
- public agm.servidor.habitaciones.HabitacionPK getPrimaryKey()
- public int getAlto()
- public void setAlto(int alto)
- public int getAncho()
- public void setAncho(int ancho)
- public java.lang.String getIdHab()
- public void setIdHab(java.lang.String idHab)
- public boolean isLuz()
- public void setLuz(boolean luz)
- public java.lang.String getMascara()
- public void setMascara(java.lang.String mascara)
- public java.lang.String getNombre()
- public void setNombre(java.lang.String nombre)
- public java.util.Collection getPuertas()
- public void setPuertas(java.util.Collection puertas)
- public java.util.List getObjetosNoPersonaje()

- `public void setObjetosNoPersonaje(java.util.List objetosNoPersonaje)`
- `public java.util.Collection getPersonajesJ()`
- `public void setPersonajesJ(java.util.Collection personajesJ)`
- `public java.util.Collection getPersonajesNJ()`
- `public void setPersonajesNJ(java.util.Collection personajesNJ)`
- `public String toString()`
- `public boolean equals(Object pOther)`
- `public int hashCode()`

agm.servidor.kernel

OyenteBean:

Es un Message Driven Bean que escucha en la cola COLA_KERNEL las peticiones de creación y conexión de personajes. Tras procesar las peticiones envía las confirmaciones por el Topic CONFIRMACION.

Atributos:

- `private javax.ejb.MessageDrivenContext messageContext`

Métodos:

- `public void setMessageDrivenContext(javax.ejb.MessageDrivenContext messageContext)`
- `public void ejbCreate()`
- `public void ejbRemove()`
- `public void onMessage(javax.jms.Message message)`
- `private void procesa(javax.jms.Message message)`
- `private TopicSession creaTopicSession()`
- `private void procesaSolicitudDesconexion(Message message)`
- `private void procesaSolicitudConexion(Message message)`
- `private void procesaSolicitudCreacionPJ(Message message)`

GestorSistemaBean:

Es un bean de sesión cuyo cometido principal es inicializar el mundo del juego, creando todas las habitaciones, objetos y personajes no jugadores, así como la cola COLA_KERNEL y el topic CONFIRMACION. Es una clase abstracta.

Atributos:

- private javax.sql.DataSource jdbcFactory
- private java.sql.Connection conn
- protected SessionContext ctx
- private RMIAdaptor server
- private HabitacionHome habitacionHome
- private KernelHome kernelHome
- private Kernel kernel
- private int contObjetos
- private int contPNJ

Métodos:

- public void setSessionContext(javax.ejb.SessionContext ctx)
- public void unsetSessionContext()
- public void ejbCreate ()
- public void borraMundo()
- private void invocaServicioMBean(String servicio, String nombreMetodo, String[] tiposParametros, Object[] valorParametros)
- protected void setReferInteg(boolean ref)
- public void setAutocommit(boolean ref)
- public void creaTablas()
- public void borraTablas()
- public void creaMundo()
- private void crearHabitacion(String idHab,String nombre,boolean luz,int ancho,int alto,String mascara,LinkedList puertas,LinkedList objetos,LinkedList pnjs)
- private void crearCasa()
- private void crearHall()
- private void crearPasillo()
- private void crearBaño()
- private void crearConserjeria()

- `private void crearDespachoX()`

GestorSistema:

Es la interfaz remota del bean GestorSistema.

Métodos:

- `public void borraMundo()`
- `public void creaTablas()`
- `public void borraTablas()`
- `public void creaMundo()`

GestorSistemaHome:

Es la interfaz home del bean GestorSistema.

Atributos:

- `public static final String COMP_NAME`
- `public static final String JNDI_NAME`

Métodos:

- `public agm.servidor.kernel.GestorSistema create()`

GestorSistemaLocal:

Es la interfaz local del bean GestorSistema.

Métodos:

- `public void borraMundo()`
- `public void creaTablas()`
- `public void borraTablas()`
- `public void creaMundo()`

GestorSistemaLocalHome:

Es la interfaz home local del bean GestorSistema.

Atributos:

- public static final String COMP_NAME
- public static final String JNDI_NAME

Métodos:

- public agm.servidor.kernel.GestorSistemaLocal create()

GestorSistemaSession:

Es una clase que hereda de GestorSistemaBean.

Atributos:

- private static agm.servidor.kernel.GestorSistemaDAO dao

Métodos:

- public void ejbActivate()
- public void ejbPassivate()
- public void setSessionContext(javax.ejb.SessionContext ctx)
- public void unsetSessionContext()
- public void ejbRemove()
- protected static synchronized agm.servidor.kernel.GestorSistemaDAO getDao()
- public void setReferInteg(boolean ref)
- public void setAutocommit(boolean ref)
- public void creaTablas()
- public void borraTablas()

GestorSistemaDAO:

Es la interfaz que contiene los *Métodos* de acceso a la base de datos para el bean GestorSistema.

Métodos:

- public void init()
- public void setReferInteg(boolean ref)
- public void setAutocommit(boolean ref)
- public void creaTablas()

- `public void borraTablas()`

GestorSistemaUtil:

Es una clase de utilidad con métodos estáticos que sirve principalmente para obtener las interfaces Home remota y local del bean GestorSistema.

Atributos:

- `private static agm.servidor.kernel.GestorSistemaHome cachedRemoteHome`
- `private static agm.servidor.kernel.GestorSistemaLocalHome cachedLocalHome`
- `private static String hexServerIP`
- `private static final SecureRandom seeder`

Métodos:

- `public static agm.servidor.kernel.GestorSistemaHome getHome()`
- `public static agm.servidor.kernel.GestorSistemaHome getHome(Hashtable environment)`
- `public static agm.servidor. kernel.GestorSistemaLocalHome getLocalHome()`
- `public static final String generateGUID(Object o)`
- `private static int getInt(byte bytes[])`
- `private static String hexFormat(int i, int j)`

KernelBean:

Es un bean de sesión cuya función principal crear personajes jugadores.

Atributos:

- `private int numJugadores`
- `private Collection personajesJConectado`
- `private Collection personajesJDesconectados`
- `private LinkedList clavesLibres`
- `private Collection claves`

Métodos:

- `public boolean idLibre(String id)`
- `public String creaPersonajeJ(String idPJ,String nombre, String password, String imagen)`

- public String loginUser (String username, String password)
- public String unloginUser (String username)
- public Collection consiguePJConectados()
- public Collection consiguePJDesconectados()
- public Collection getPersonajesJConectados()
- public Collection getPersonajesJDesconectados()
- public int getNumJugadores()
- public void setPersonajesJConectados(Collection collection)
- public void setPersonajesJDesconectados(Collection collection)
- public void setNumJugadores(int i)
- public Kernel ejbCreate()
- public Collection ejbHomegetInventarioFijo()

Kernel:

Es la interfaz remota del bean Kernel.

Métodos:

- public java.lang.String creaPersonajeJ(java.lang.String idPJ,java.lang.String nombre,java.lang.String password,java.lang.String imagen)
- public java.lang.String loginUser(java.lang.String username,java.lang.String password)
- public java.lang.String unloginUser(java.lang.String username)
- public java.util.Collection consiguePJConectados()
- public java.util.Collection consiguePJDesconectados()
- public java.util.Collection getPersonajesJConectados()
- public java.util.Collection getPersonajesJDesconectados()
- public int getNumJugadores()
- public void setPersonajesJConectados(java.util.Collection collection)
- public void setPersonajesJDesconectados(java.util.Collection collection)
- public void setNumJugadores(int i)

KernelHome:

Es la interfaz home del bean Kernel.

Atributos:

- public static final String COMP_NAME
- public static final String JNDI_NAME

Métodos:

- public agm.servidor.kernel.Kernel create()

KernelLocal:

Es la interfaz local del bean Kernel.

Métodos:

- public java.lang.String creaPersonajeJ(java.lang.String idPJ,java.lang.String nombre,java.lang.String password,java.lang.String imagen)
- public java.util.Collection getPersonajesJConectados()
- public java.util.Collection getPersonajesJDesconectados()
- public int getNumJugadores()
- public void setPersonajesJConectados(java.util.Collection collection)
- public void setPersonajesJDesconectados(java.util.Collection collection)
- public void setNumJugadores(int i)

KernelLocalHome:

Es la interfaz home local del bean Kernel.

Atributos:

- public static final String COMP_NAME
- public static final String JNDI_NAME

Métodos:

- public agm.servidor.kernel.KernelLocal create()

KernelSession:

Es una clase que hereda de KernelBean.

Atributos:

- private static agm.servidor.kernel.KernelDAO dao

Métodos:

- public void ejbActivate()
- public void ejbPassivate()
- public void setSessionContext(javax.ejb.SessionContext ctx)
- public void unsetSessionContext()
- public void ejbRemove()
- protected static synchronized agm.servidor.kernel.KernelDAO getDao()
- public java.lang.String loginUser(java.lang.String username,java.lang.String password)
- public java.lang.String unloginUser(java.lang.String username)

KernelDAO:

Es la interfaz que contiene los *Métodos* de acceso a la base.de datos para el bean Kernel.

Métodos:

- public void init()
- public java.lang.String loginUser(java.lang.String username,java.lang.String password)
- public java.lang.String unloginUser(java.lang.String username)

KernelUtil:

Es un una clase de utilidad con métodos estáticos que sirve principalmente para obtener las interfaces Home remota y local del bean Kernel.

Atributos:

- private static agm.servidor.kernel.KernelHome cachedRemoteHome
- private static agm.servidor.kernel.KernelLocalHome cachedLocalHome
- private static String hexServerIP
- private static final SecureRandom seeder

Métodos:

- public static agm.servidor.kernel.KernelHome getHome()
- public static agm.servidor.kernel.KernelHome getHome(Hashtable environment)

- `public static agm.servidor. kernel.KernelLocalHome getLocalHome()`
- `public static final String generateGUID(Object o)`
- `private static int getInt(byte bytes[])`
- `private static String hexFormat(int i, int j)`

PAQUETE: agm.dao

KernelDaoImpl:

Es la clase que contiene los *Métodos* que usa el bean Kernel para acceder a la base de datos.

Atributos:

- `private DataSource jdbcFactory;`

Métodos:

- `public void init()`
- `public String loginUser(String userId, String password)`
- `public String unloginUser(String userId)`

GestorSistemaDaoImpl:

Es la clase que contiene los *Métodos* que usa el bean GestorSistema para acceder a la base de datos.

Atributos:

- `private DataSource jdbcFactory`
- `private Connection conn`

Métodos:

- `public void init()`
- `public void setReferInteg(boolean ref)`
- `public void setAutocommit(boolean ref)`
- `public void creaTablas()`
- `private void executeQuery(PreparedStatement ps)`
- `public void borraTablas()`

HabitacionDaolmpl:

Es la clase que contiene los *Métodos* que usa el bean Habitacion para acceder a la base de datos.

Atributos:

- private DataSource jdbcFactory

Métodos:

- public void init()
- private void loadObjetosNoPersonaje(java.sql.Connection conn, HabitacionPK pk, Collection objetosEnHab, Collection puertas)
- private LinkedList loadInventario(java.sql.Connection conn, String idP)
- private agm.servidor.kernel.KernelHome getKernelHome()
- private Collection loadPersonajesJugadores(java.sql.Connection conn, HabitacionPK pk)
- private static InitialContext getContext()
- public Collection loadPersonajesNoJugadores(java.sql.Connection conn, HabitacionPK pk)
- private TopicSession creaTopicSession()
- public void load(HabitacionPK pk, HabitacionBean ejb)
- private void borraContenidoHabitacion(java.sql.Connection conn, HabitacionPK pk)
- private void storePersonajesNJ(java.sql.Connection conn, HabitacionBean ejb)
- private void storeObjetosNoPersonaje (java.sql.Connection conn, HabitacionBean ejb)
- private void storePersonajesJ(java.sql.Connection conn, HabitacionBean ejb)
- public void actualizaAtributosSimplesDeHabitacion(java.sql.Connection conn, HabitacionBean ejb)
- public void store(HabitacionBean ejb)
- public HabitacionPK findByIdP(String idP)
- public HabitacionPK findByPrimaryKey(HabitacionPK pk)
- public void remove(HabitacionPK pk)
- public HabitacionPK create(HabitacionBean ejb)
- public HabitacionPK findByIdPNJ(String idPNJ)
- public HabitacionPK findByIdPJ(String idPJ)

- `public HabitacionPK findByIdO(String idO)`
- `public Collection findAll()`
- `public Collection findByNombre(String nombre)`
- `public String getClaveLibreParaObjetoNP()`

Aventura Gráfica FDI Multijugador

PRUEBAS

1. Diseño de las pruebas realizadas

Para probar las pruebas adjuntas a este documento, hay que ejecutar "Prueba.java" que es quien tiene el método main. La versión del prototipo de pruebas es ligeramente diferente a la implementación final.

Las pruebas realizadas son las siguientes:

Nombre de la prueba	0
Objetivo de la prueba	<i>Prueba de creación de personajes jugadores simultánea</i>
Breve descripción	<p>Su funcionamiento es el siguiente, mediante el uso de una interfaz textual, se pide por consola el número de personajes jugadores que queremos tratar de crear simultáneamente.</p> <p>Se iterará tantas veces como jugadores se indicó anteriormente, y en cada iteración se creará un hilo <i>PruebaCreacionPJ</i>, cada uno de estos hilos crea un <i>ClienteJugador</i> y comienza una espera activa.</p> <p>Cuando el último de los hilos ha sido creado, el hilo padre activa un flag que da permiso a todos los hilos hijo para invocar el método <i>solicitudCreacionPJ</i> con unos valores de jugadores generados secuencialmente</p>
Programa de pruebas	pruebas.Prueba
Arranque del programa	Desde el método main() de pruebas.Prueba
Éxito de la prueba	Se han lanzado hasta 100 solicitudes de creación de personajes jugadores simultáneamente y con éxito.

Nombre de la prueba	1
Objetivo de la prueba	<i>Prueba de conexión de personajes jugadores simultánea</i>
Breve descripción	<p>Su funcionamiento es el siguiente, tras lanzar la prueba a, invoca a <i>solicitudConexion</i> de todos los <i>ClienteJugador</i> que han creado personajes. No se crea una interfaz gráfica, como se mencionó anteriormente.</p>
Programa de pruebas	pruebas.Prueba
Arranque del programa	Desde el método main() de pruebas.Prueba
Éxito de la prueba	Se han lanzado hasta 10 solicitudes de creación de personajes jugadores simultáneamente y con éxito.

Nombre de la prueba	2
Objetivo de la prueba	<i>Prueba de cambio de habitación simultáneo.</i>
Breve descripción	Su funcionamiento es el siguiente, tras lanzar la prueba b, invoca a ejecutarAccion del cliente jugador pasándole como parámetros el id de la puerta que lleva al baño y el id de la acción "Ir A". Se trata al objetoRefresco que se recibe como respuesta de la acción de igual forma que lo hace la aplicación normalmente, salvo que no se invoca a ningún tipo de refresco, ya que no tenemos ninguna interfaz gráfica.
Modo de prueba	Mediante jugadores reales
Éxito de la prueba	3 jugadores han podido pasar a sus personajes de una habitación a otra a través de la misma puerta y haciendo sus peticiones de atravesar la puerta simultáneamente..

Nombre de la prueba	3
Objetivo de la prueba	<i>Prueba de ejecutar acción que modifica el inventario simultáneamente.</i>
Breve descripción	Su funcionamiento es el siguiente, tras lanzar la prueba c, invoca a a ejecutarAccion del cliente jugador pasándole como parámetros el id del objeto pastillero y el id de acción "Coger".
Modo de prueba	Mediante jugadores reales
Éxito de la prueba	3 jugadores han podido realizar su petición de coger jabón simultáneamente y con éxito.

Historias de análisis

Sistema transmisor de conocimientos

Personajes

PJ's:

Protagonistas

- B: es alumno del profesor X y no tiene aprobada la asignatura que X imparte.

Colaboradores

- C: Será el encargado de encender la máquina una vez que el profesor y el alumno B estén conectados a la máquina.
- A: se disfraza de cocinero y pone polvos somníferos en la comida del profesor X para dormirle.

PNJ's:

- Profesor X: imparte una asignatura que A no ha aprobado aún.

Objetos:

- Sistema transmisor de conocimientos: consiste en una máquina que se conecta a dos personajes. Su función es transmitir el conocimiento sobre una asignatura concreta de un personaje a otro. La conexión del personaje que va a aprender la realizan unos cables que van de su cerebro directamente a la máquina. La conexión del personaje que aporta sus conocimientos se realiza remotamente. Para ello un radiotransmisor se conecta con cables al cerebro del personaje y al otro lado la máquina recibe la información transmitida por el radiotransmisor en forma de ondas.

Flujo de sucesos

1. A acude a la tienda de la facultad.
2. El sistema muestra un menú con los posibles comercios.
3. A elige el comercio boutique
4. Se le muestra un menú con los productos disponibles
5. A elige un disfraz de cocinero y selecciona la opción comprar pasando a formar parte de su inventario el disfraz y disminuyendo su cantidad de dinero en lo que vale el producto comprado.
6. A sale de la boutique
7. A entra a uno de los servicios
8. A usa el disfraz
9. A sale del servicio
10. A entra a la cafetería a la hora de la comida
11. A intenta entrar en la cocina a través de la puerta que comunica la cafetería con la cocina.

Casos:

- Éxito: no es descubierto porque va disfrazado. Ir a 12.
- Fracaso: es descubierto por un PNJ cocinero
 1. El PNJ cocinero envía a A directamente a la cárcel
 2. Casos:
 - a. A paga la fianza con su propio dinero. Ir a 12.
 - b. A no paga y por tanto no sale de la cárcel.

Nota: Si A no dispone del dinero para pagar la fianza, A deberá esperar el tiempo suficiente que le permita obtenerlo a través de las pagas.

12. A se dirige hacia la encimera donde el cocinero auténtico deja los platos preparados para ser servidos.

13. A busca entre los platos preparados para ser servidos un plato etiquetado con el nombre del profesor X.
14. A deberá esperar a que no haya ningún cocinero en la cocina si no quiere que le descubran. Entonces A intenta usar los polvos somníferos sobre la comida del profesor. Casos:
 - Fracaso: si hay otro cocinero en ese momento dentro de la cocina, le envía a la cárcel y solo podrá salir de aquí por el mismo procedimiento que antes.
 - Éxito: si no hay ningún otro cocinero en el momento que echa los polvos sobre la comida del profesor. Ir a 15.
15. A se dirige al comedor y busca una mesa donde esté comiendo el profesor o ponga una etiqueta de reserva a su nombre. Casos:
 - Si el profesor todavía no está en la mesa, entonces: lo espera.
 - Si el profesor ya está comiendo, entonces espera a que esté dormido.
 - Si el profesor está dormido, entonces usa el sistema radio transmisor de conocimientos sobre la cabeza del profesor.
16. Mientras tanto, B debe entrar al departamento del DACYA.
17. Debe buscar unos cables y cogerlos
18. Debe buscar el sistema transmisor de conocimientos y usar los cables sobre la cabeza primero y luego sobre el sistema para establecer conexión entre cabeza y sistema.
19. Una vez que el profesor y el alumno B estén conectados a la máquina, C activará el interruptor del sistema.

Electrocución en el panel de botones del ascensor

Personajes

PJ's:

Protagonistas

A: manipula los cables del panel de botones del ascensor para que los personajes que pulsen uno de los botones del ascensor queden electrocutados durante un tiempo de 1 minuto.

Colaboradores

B: desactiva la corriente para que A no se electrocute

C: alumbra con una linterna para que A pueda manipular el panel de botones del ascensor.

PNJ's:

- Guardia de seguridad F: vigila el garaje permanentemente.
- Guardia de seguridad G: está dentro del ascensor vigiándolo

Flujo de sucesos

1. Un personaje B se dirige a la sala de control de la facultad
2. Opcional: se acerca a una ventanilla de la sala de control que da al pasillo, levanta la cortinilla que hay sobre esta ventanilla y mira para ver si está dentro el encargado del sistema de control
3. Casos:
 - B espera a que el encargado del puesto de control salga de la sal.
 - B intenta entrar pero no puede porque el encargado se lo impide.
 - Si el encargado ha salido para ser sustituido por el otro encargado, B puede entrar.
4. Solo cuando B ha entrado en la sala, buscará el interruptor en cuestión
 - 1.4.1 Caso de éxito:
 - 1.4.1.1 Cierra el interruptor que regula la entrada de corriente al motor del ascensor y a las luces que lo alumbran para evitar que A se electrocute.
 - 1.4.1.2 Sale de la sala
 - 1.4.2 Caso de fracaso:
 - 1.4.2.1 Durante el tiempo que B está dentro de la sala, el sustituto entra en la sala
 - 1.4.2.2 Una vez dentro, el sustituto advierte a B de que está en un sitio restringido a solo personal autorizado y como multa le quita el dinero que lleva encima. Además envía a B al cuartelillo.
 - 1.4.2.3 B avisa por teléfono móvil a un personaje cualquiera E para que le saque de la cárcel.
 - 1.4.2.4 E debe
 - 1.4.2.4.1 Pedir una llave maestra en la conserjería
 - 1.4.2.4.2 Ir al piso que está encima de la planta de la cárcel
 - 1.4.2.4.3 Localizar una tapa camuflada bajo una maceta que hay en el suelo y que da acceso al piso inferior justo en el techo de la celda donde está B.
 - 1.4.2.4.4 Abrir la cerradura con la llave maestra
 - 1.4.2.5 El personaje B sitúa una silla que tiene en la celda en la vertical con la apertura que ha quedado en el techo
 - 1.4.2.6 Se sube a la silla.
 - 1.4.2.7 Sale de la celda a través del hueco.
5. Un personaje D debe ir al garaje y entablar una conversación con el guardia de seguridad que está dentro del garaje.
6. B se dirige a donde están D y el guardia de seguridad y aprovechando que está distraído hablando, debe robarle las esposas al guardia.
7. El guardia F que cuida el garaje solicita ayuda al guardia de seguridad G que cuida el ascensor para poder detener al ladrón. Como consecuencia, G se dirigirá al garaje.

8. El personaje C alumbra con una linterna dentro del ascensor mientras tanto A manipula los cables.
9. Casos:
 - Éxito: A termina la operación sin ser descubierto por ningún guardia de seguridad. Por haber tenido éxito, A y sus colaboradores se llevan una recompensa a elegir en dinero o en puntos de estudio.
 - Fracaso: si A intenta manipular el panel de botones del ascensor cuando el guardia de seguridad G todavía no ha partido hacia el garaje, G le detendrá y le llevará al cuartelillo donde deberá permanecer por un tiempo de 2 minutos y pagará una multa consistente en todo el dinero del que dispone, es decir, tanto el que lleva encima, como el que guarda en su taquilla.

Chantaje a un profesor del SIP robando un examen para luego conseguir dinero

Personajes

PJ's:

Protagonistas

A: alumno

PNJ's:

- Profesor del SIP(B)

Definición del flujo de la historia desde el punto de vista de las acciones primitivas llevadas a cabo por el jugador que dirige a su personaje jugador para completar la historia.

Cuando se crea un personaje mostrar:

Tengo un examen de EDI y debo ir a tutorías para preguntar al profesor como se resuelve un ejercicio que entró en el examen del año anterior y que este año también va a entrar.

1. A hace IR_A sobre puerta del despacho del profesor

Efecto: meter al alumno A en el despacho del profesor.

Mostrar: ya estoy dentro del despacho del profesor.

2. A HABLA CON profesor

Mostrar:

A: "Hola profe, ¿me ayudas a resolver este problema del examen del año pasado y que nos dijiste que también iba a entra en el examen de este año?"

B: "No me hables, no me hables pues mi novia me ha dejado y quiero estar solo bebiendo y bebiendo a no ser que tengas algo de whisky, porque se me ha acabado y lo necesito urgentemente para quitarme las penas."

A : "Tengo una idea: si me das una copia de las respuestas del próximo examen de EDI, te doy una botella de whisky."

B : "Pensándolo mejor voy a ser educado contigo y acepto tu propuesta pero antes debes darme tu la botella de whisky."

Efecto: poner una marca en el estado del personaje jugador para saber que por lo menos ha hablado una vez con el profesor.

3. A USA botella de whisky del inventario sobre profesor

Mostrar:

B: "no te voy a dar la contraseña de mi pc pues aunque esté algo ebrio todavía no lo estoy tanto como para darte acceso a mi pc. Sin embargo como lo prometido es deuda, te diré la contraseña del candado que cierra el cajón de mi despacho donde guardo una copia

de seguridad en diskette con las respuestas del próximo examen. La contraseña del candado es "374".

A: "He perdido la botella de whisky pero ha cambiado he conseguido la contraseña. Debería volver al despacho del profesor y coger el disquete."

Efecto: Sacar la botella de whisky del inventario de A.

Curso alterno:

- A nunca antes había hablado con el profesor y A USA botella de whisky CON profesor

Mostrar:

B: "No me intentes dar un botellazo con la botella de whisky que te veo venir."

Efecto: ninguno.

4. A hace IR_A sobre la salida de ventilación del despacho del profesor para entrar al despacho ya que la puerta está siempre cerrada con llave.

Efecto: meter al personaje en el despacho del profesor.

Mostrar: He conseguido entrar al despacho del profesor de un modo poco ortodoxo, pero lo he conseguido. Ahora ya solo me queda localizar el disquete.

Cursos alternos:

- IR_A sobre la puerta del despacho:

Mostrar: "Necesito la llave del despacho pues la puerta está cerrada."

Efecto: Ninguno.

- USA llave sobre puerta del despacho:

Mostrar:

○ "Esta cerradura no hay quien la abra ni con la llave..."

○ "Bah, vaya mala suerte, la llave se me ha roto mientras intento abrir la puerta."

- USA martillo sobre puerta

Mostrar: "Esta puerta está blindada y no conseguiré tirarla abajo."

Efecto: ninguno.

5. A USA candado del cajón del despacho

Mostrar: cuadro de diálogo pidiendo contraseña

6. A mete contraseña "374"

Mostrar: "He abierto el cajón."

Efecto: cambiar el estado del despacho de "CERRADO" A "ABIERTO"

7. A COGE disquete

(A sale del despacho y va al laboratorio)

8. A USA disquete sobre impresora.

Mostrar: "Ya tengo una chuleta impresa con las respuestas del examen sin embargo parece que la impresora no quiere devolverme mi disquete. ¡será ladrona la impresora!".

Efecto: Meter al inventario de A "chuleta" con las respuestas del examen.

Sacar del inventario de A el disquete pues se lo ha quedado la impresora.

(A va al aula donde se celebra el examen de EDI y entra al aula)

9. A HABLA con examinador

Mostrar:

A: vengo a hacer un examen.

Examinador: ten una copia del enunciado.
Efecto: Meter un “examen” en el inventario del personaje.

10. A USA el elemento del inventario llamado “chuleta” CON el elemento del inventario “examen”.

Mostrar: He obtenido un sobresaliente en EDI.

Efecto:

1. Sacar la hoja de enunciados y la chuleta del inventario del personaje.
2. Meter al expediente del personaje un sobresaliente en EDI.

Descubrir el pasadizo que lleva a la facultad de Derecho

Personajes

PJ's:

Protagonistas

A,B,C: alumnos

PNJ's:

Decana

Bibliotecario

Flujo de sucesos

Introducción: un camión ha llegado a la facultad con muchos documentos para la decana que están siendo descargados por un operario que además los introduce en carritos. Estos carritos son llevados por otro operario al despacho de la decana que está muy ocupada resolviendo el papeleo relacionado con los documentos que le están llegando y por ello en este momento no admite visitas. A quiere hablar con la decana para preguntarle que dónde está el pasadizo secreto que permite llegar desde la facultad de informática a la facultad de Derecho. Mientras tanto la única oportunidad que tiene el personaje A para colarse en el despacho de la decana es dirigirse a la puerta que da acceso al garaje.

1. A USA martillo CON cerradura

Efecto: meter al personaje dentro del garaje

Mostrar: Parece que he conseguido entrar al garaje forzando la cerradura

Una vez dentro del garaje, A buscará el camión que está siendo descargado. También buscará uno de los carritos pendientes de ser llevados al despacho de la decana.

2. A USA rotulador CON carrito

3. A USA carrito

Lo siguiente se hace en secuencia:

Efecto1: meter a A a CASA durante unos segundos

Poner en negro la pantalla del jugador.

Mostrar1:

“No veo nada. Ah... creo que me he metido dentro de uno de los carritos que van a ser llevados al despacho de la decana”

Efecto2: sacar a A de CASA y meterlo en el despacho de la decana.

Mostrar2: "Estoy saliendo del carrito y me encuentro en un sitio desconocido, aunque esa cara me suena. Diría que es la decana. Ah, ya está me he debido meter en el despacho de la decana en uno de los carritos que tras ser descargados del camión fueron llevados a su despacho."

Cursos alternos:

- A no había marcado previamente el carrito con un rotulador.
Efecto: ninguno
Mostrar: "Si me meto en el carrito, luego no podrán sacarme pues hay tantos carritos que no sabrán en cual estoy".

Mientras tanto un tercer personaje C estará esperando a la puerta del despacho de la decana viendo todos los carritos que el operario está introduciendo en su despacho.

4. A HABLA CON decana

Efecto: ninguno.

Mostrar:

Decana: "Pero que haces tu aquí"

A: "¿Sabes dónde está la boca del pasadizo?"

Decana: "¿De que pasadizo me estás hablando?"

A: "Si hombre, si, si creo que en el subsuelo de esta facultad hay pasadizos a una profundidad igual a la altura del edificio."

Decana: "Bueno, bueno, parece que estás enterado. Te diré una cosa. Te voy a contar un secreto. La boca del pasadizo se encuentra oculta detrás de una estantería de la biblioteca. Pero esta estantería está trucada por el bibliotecario y solo se abate cuando las manos del bibliotecario cogen el único libro que no tiene título de toda la biblioteca y que por tanto no figura en el catálogo de la misma. Este libro está en la misma estantería que cubre la boca del pasadizo. Ahora bien, si quieres que el bibliotecario os busque el libro sin título, debes darle la contraseña "984"

Entonces A irá a la biblioteca.

5. A HABLA CON bibliotecario

Efecto: ninguno.

Mostrar:

Bibliotecario: "Hola buenas tardes. ¿Qué deseas?"

A: "Deseo que me des de el único libro sin título que hay en la biblioteca."

Bibliotecario: "Creo que sé de que libro me estás hablando, pero hay un problema y es que necesito que me des una contraseña porque ese libro no es de préstamo público sino que está restringido a muy pocas personas."

A: Si una contraseña te sirve podría intentar darte una.

Biblioteca: Venga dámela.

6. A USA papel CON bibliotecario

Mostrar:

Bibliotecario: "Ahora mismo te doy el libro."

A: "El bibliotecario se está dirigiendo a la librería de la esquina y parece que está sacando el libro de la misma."

A se dirige a la librería que se ha abierto y ha dejado al descubierto un hueco en la pared.

7. A hace IR_A librería

Efecto1: meter a A a la facultad de derecho

Mostrar1: "Me he metido en la facultad de derecho aunque poco puedo hacer aquí. Aquí no conozco a nadie. Mejor será que vuelva a la facultad de informática."

Tras unos segundos durante los cuales el personaje A permanece en la facultad de derecho

Efecto2: meter a A al HALL de la facultad de informática

Mostrar2: "Ya he vuelto a la facultad de informática."

Aprobado de máximo riesgo

Un jugador sube a la azotea, y cambia el aire acondicionado de los despachos. Esto hace que un profesor PNJ, cuando vaya a su despacho, tenga calor y abra la ventana.

El mismo jugador ha cogido un juego de manteles de un carrito de los empleados de la limpieza, y desde la misma azotea ata los manteles en forma de cuerda y se desliza hacia abajo, hasta la ventana del profesor.

Ahora bien, desde la ventana puede coger una copia del examen, siempre que este no mire. Puede no mirar si recibe una visita (tutoría) de otro jugador que en ese momento desee hablar con el (ese jugador no tiene porque estar cooperando).

El jugador también ha podido conseguir el número de teléfono del despacho y llamar al profesor con su móvil, para distraerle y coger el examen, y volver a descolgarse hacia arriba.

NOTAS: cada cierto tiempo, el aire acondicionado es restaurado. Su modificación no afecta a otros escenarios. Los manteles se reponen cada cierto tiempo, y a esos efectos, son infinitos.

Unas botellitas... y un jamón

Pablo Gervás como PNJ, esta principalmente en uno de los laboratorios. Una vez allí si el jugador hace que le llamen o recibe una llamada, Gervás le impondrá una botella de cava por la impertinencia.

Dicho profesor, pasará también varias veces por conserjería, donde hay cerca un tipo de esos que te hacen fotos con una polaroid por un módico precio. Cuando esté allí el jugador puede pedirle hacerse una foto con él. Con esa foto, en la tienda puede conseguir una máscara (pierde la foto).

Con la máscara se podrá entrar al comedor de profesores y desde allí a la cocina, donde se podrá coger un jamón.

10. Otra forma de coger el jamón: esconderse en un carrito de bandejas de comida, y esperar a que te lleven a la cocina.
11. La botella de cava o bien se puede comprar, o en el escenario del disfraz de camarero, se puede pasar detrás de la barra y coger una.

Cuando tengas estos dos objetos, podrás llevárselos a Gervás. Primero la botella, y luego si el jamón. No podrá resistirse a tus dotes con la IS y la forma de presentar "un trabajo" y te dará el aprobado.

¿Aquellos maravillosos años?

Un jugador, hablando con un profesor, por ejemplo Pedro (el de EDI), escucha como este se enorgullece de que repite un examen cada 10 años para trabajar menos.

Después de hablar sobre eso, puedes ir a la biblioteca, y buscar (un objeto especial puede hacer más fácil tu búsqueda) una tesis super escondida y olvidada: "Como diseñar un 286 con tan solo 4 resistencias y 3 transistores".

Copiando lo esencial en una hoja (se debe tener papel, o bien se compra o se manga de una impresora), el jugador puede ir al laboratorio 8.

Con un objeto tal como el fémur que recibes del fantasma del sótano, puede abrir el armario que da a los cacharros, y coger las resistencias y transistores, colocarlas en el entrenador, darle a la corriente y... PLAF, un vórtice espacio tiempo le manda a la facultad del pasado.

Aquí entra en una historia monojugador, llana y sencilla, con el único objetivo de que los nuevos jugadores vean algunas cosillas de la antigua facultad (o escuela) y en cuanto se coja la hoja de examen (que 10 años en el futuro, en su tiempo, será repetido) puede volver a su tiempo por el mismo agujero. Al volver las resistencias están quemadas e inservibles. La hoja donde anotó las cosas se ha convertido en una rama de árbol. La tesis no se puede sacar de la biblioteca.

La puerta del armario de los cacharros está rota, pero con el tiempo, algún PNJ puede arreglarla. Si antes del arreglo un jugador se la encuentra abierta, esa suerte que tiene (o no).

El tesoro escondido

En algún lugar oculto de la facultad hay un tesoro escondido. Encontrándolo se conseguirá el aprobado para alguna de las asignaturas más difíciles de la carrera (por ejemplo MTP). Además, escondido junto al tesoro hay un objeto mágico único, que proporcionará al jugador que lo coja y lo entregue a determinado PNJ la matrícula de honor.

El jugador antes de nada deberá encontrar el mapa del tesoro en el que se indica el lugar del edificio en el que está la entrada oculta a la habitación secreta, en la cual se encuentra el tesoro. Dicho mapa se encuentra en un ordenador que hay dentro de un despacho cerrado. El jugador deberá sobornar al conserje para que le diga que habitación es y cual es la contraseña del ordenador. La contraseña será temporal y personal (sólo podrá utilizarla dicho jugador y durante un espacio de tiempo establecido).

Como hay un guardia vigilando el pasillo donde está el despacho, el jugador deberá dormirle con unos polvos somníferos (o similar) mientras otro jugador habla con él o mientras le llaman por teléfono. Una vez que el guardia está dormido hay que forzar la puerta del despacho, o abrirla con llave si es que se ha conseguido una copia. Dentro el jugador deberá introducir la contraseña en el ordenador y podrá ver el mapa, que deberá copiarse en una hoja, ya que no hay impresora.

Cuando tenga el mapa, debe ir a la biblioteca a buscar un libro escondido donde encontrará el nombre del PNJ al que debe entregar el objeto mágico para conseguir la matrícula, en caso de que nadie lo haya cogido aún de su escondite. Para encontrar el libro necesita la ayuda del bibliotecario, y éste sólo le ayudará a cambio de su comida favorita. Por lo tanto habrá que averiguar cual es y conseguirla en la cafetería o en una máquina.

Una vez hecho esto, el jugador puede dirigirse al lugar indicado en el plano (por ejemplo el garaje), donde habrá una puerta oculta, que deberá encontrar y abrir, con ayuda de una palanca por ejemplo. Cuando la puerta esté abierta necesitará utilizar una linterna o una vela, ya que la habitación secreta se encuentra a oscuras. Allí dentro habrá un cofre que contiene el tesoro; cuando el jugador consiga abrirlo (con el fémur del fantasma o algún otro utensilio) ya estará automáticamente aprobado y recibirá una cantidad de dinero como premio.

Puede ser que al lado del objeto haya una cajita con el objeto mágico, o bien porque es el primero que encuentra el tesoro, o porque los que llegaron anteriormente no se lo llevaron. Si coge la

cajita y en el plazo de 5 minutos se la entrega al PNJ cuyo nombre encontró en el libro, será recompensado con una MH.

Reunión de profesores

Se va a celebrar una reunión de profesores donde se van a decidir las preguntas de uno o varios exámenes. Un jugador alumno podría conseguir una copia de dicho examen si estuviera presente, pero sólo se permite la entrada a los profesores, así que el jugador tendrá que hacerse pasar por un profesor para poder entrar.

El jugador deberá sobornar al conserje para que le dé el nombre de un profesor que se emborrache fácilmente y cual es su marca de licor preferida. Cuando sepa esto debe conseguir una botella de dicha marca de la cocina, disfrazándose de camarero o cocinero, o comprándola en una tienda, aunque será en un objeto muy, muy caro, debido a los beneficios que se pueden obtener utilizándola.

Después deberá disfrazarse con un gorro y unas gafas de sol para ir a hablar con dicho profesor, para que posteriormente no pueda reconocerlo, lo cual tendría graves consecuencias para el jugador. Cuando esté en el despacho le ofrecerá la botella al profesor, que se la beberá entera casi inmediatamente y acabará tan borracho que no se dará cuenta de lo que pasa a su alrededor. Entonces el jugador podrá buscar una llave del despacho y salir de allí dejando encerrado al profesor.

Entonces puede ponerse una máscara con la cara del profesor (que tiene que haber conseguido previamente) y dirigirse al despacho donde tiene lugar la reunión. Debe llevar una grabadora encima para poder recordar las preguntas del examen ya que no puede escribirlas en un papel delante de los demás profesores. Justo antes de entrar debe poner en marcha la grabadora, ya que no tendrá ocasión de hacerlo una vez que esté dentro.

En la reunión se enterará de las preguntas del examen, que deberá copiar después en un papel para poder ir a la biblioteca o a un ordenador a buscar las respuestas. Cuando llegue el momento del examen entregará su chuleta y tendrá un 10.

SOLUCIONES A PROBLEMAS CON LOMBOZ

Como integrar código de distintos proyectos lomboz

Para lo siguiente se supone que ya tenemos instalado el plugin del Lomboz para eclipse tal y como describe el capítulo 1 del tutorial del Lomboz. Pasos a realizar:

- 1) File → New LombozJ2EEProject
Creamos un proyecto llamado Integracion.
- 2) File → new → LombozJ2Eemodule
Creamos un EJBModule Prototipo y ponemos como target server el nombre asociado al JBoss que tengamos instalado seleccionándolo de la lista *type* que se muestra.
- 3) Metemos en el directorio src del proyecto Integracion el contenido de todos los directorios src que estamos integrando.
- 4) Window → Show View → Package Explorer
Una vez abierto el package explorer desplegamos en el icono Integracion que aparece en el package explorer. Luego desplegamos src, y luego vamos desplegando cada uno de los paquetes de que se componga el código. Ahora sobre cada uno de los ficheros .java de cada paquete haremos click derecho
→ LombozJ2EE → Add EJB to module y como módulo seleccionamos el creado en el apartado 2), es decir, el módulo Prototipo. De esta forma añadimos estos ficheros al módulo Prototipo. Para algunos ficheros fuente .java esta opción de Add EJB to module no aparecerá disponible. Para estos no pasa nada.
- 5) Luego haremos click derecho en el + Integracion(EJB) que aparece en el package explorer (suele aparecer en la parte de abajo del package explorer) y en el menú que se nos muestra → LombozJ2EE → Generate EJB classes
- 6) Ahora ya tenemos un único proyecto integrado.

Solución a cuando tras integrar y ejecutar EJB Generate, xdoclet muestra XDoclet classpath missing J2EE classes

Es posible que el fichero .classpath del directorio del proyecto no contenga las rutas necesarias. Para que este fichero contenga las rutas necesarias hay que crear explícitamente al menos un módulo de ejbs, es decir, que no basta con meter módulo ya hecho en el directorio del proyecto sino que también es necesario crear un módulo nuevo cuando se crea un proyecto de integración. Otra posibilidad es coger el fichero .classpath de otro proyecto que ya funcione y ponerlo en lugar del que tenemos. De esta segunda forma también solucionaremos el mensaje que nos da xdoclet.

Como configurar mysql para ejecutar el cliente del capítulo 3 del tutorial del lomboz:

- 1) instalar el puente ODBC para MySql (viene en la página de <http://www.mysql.com> => Documentation → MySQL Connector/J Documentation
- 2) crear una base de datos en MySqlControlCenter
- 3) crear las tablas con los scripts que vienen en el tutorial, desde el cliente de MySql (MySqlControlCenter) sobre la base de datos que hemos creado en el apartado 2).
- 4) modificar el fichero xdoclet.xml y standardjbosscmp-jdbc.xml según se explica en el tutorial del Lomboz. Un resumen es:

Para usar el gestor de bases de datos MySQL en lugar del gestor que viene configurado por defecto con Lomboz hay que cambiar el fichero xdoclet.xml generado en meta-inf. Donde pone mySQL antes ponía Hipersonic. Entonces habrá que poner mySQL.

```
<jboss
  version="3.0"
  xmlencoding="UTF-8"
  destdir="${ejb.dd.dir}"
  validatexml="false"
    datasource="java:/DefaultDS"
  datasourcemapping="mySQL"
  preferredrelationmapping="foreign-key"
/>
```

También habrá que cambiar el fichero jbosscmp-jdbc.xml:

```
<jbosscmp-jdbc>
<defaults>
  <datasource>java:/DefaultDS</datasource>
  <datasource-mapping>mySQL</datasource-mapping>
  <preferred-relation-mapping>foreign-key</preferred-relation-mapping>
</defaults>
```

Donde pone mySQL ponía Hipersonic. Hay que cambiarlo para que ponga mySQL.

Solución al problema: cuando hacemos Generate EJB classes de beans con tags de xdoclet `@ejb.dao class`, no se generan los interfaces DAO correspondientes.

En el fichero xdoclet.xml hay que incluir la línea que aparece en negrita porque no es incluida por el EJB Generate.

```
<dataobject/>
<valueobject/>
<utilobject cacheHomes="true" includeGUID="true"/>
```

```
<dao pattern="{0}" destDir="${project.dir}/${ejbsrc.dir}" />
<remoteinterface/>
<localinterface/>
```

Solución al problema: cuando se hace un build, dirá que faltan los siguientes archivos .jar

```
/jboss/server/all/lib/jboss-transaction.jar
/jboss/lib/xml-apis.jar
/jboss/lib/xercesImpl.jar
/jboss/lib/webdavlib.jar
/jboss/lib/jdom.jar
/jboss/lib/commons-httpclient.jar
```

Estos archivos los debemos copiar desde la distribución jetty del JBoss a la distribución estándar 3.0.8

Solución al error que sale al hacer el deploy del bean del capítulo 3 del tutorial del lomboz:

In our {eclipsehome}\plugins\com.objectlearn.jdt.j2ee\servers, there's a .server file jboss321all.server, add the code below just after the "</projectClassPath>" tag in ur jboss321all.server file

```
<adminTool>
<web><deploy /><undeploy />
</web>
<ejb><deploy /><undeploy />
</ejb>
<ear><deploy /><undeploy />
<webModule><deploy /><undeploy />
</webModule>
<ejbModule><deploy /><undeploy />
</ejbModule>
</ear>
</adminTool>
<adminToolPath>
</adminToolPath>
```

Solución a bugs en la generación del jboss.xml por EJB Generate:

Consiste en que en las referencias a interfaces locales pone <ejb-ref> en lugar de poner <ejb-local-ref> y </ejb-ref> en lugar de </ejb-local-ref>

Por eso cada vez que se invoca EJB Generate hay que cambiar en el fichero jboss.xml lo que acabo de comentar.

Un ejemplo de trozo de fichero jboss.xml generado por Lomboz sería:

```
<session>
  <ejb-name>StoreAccess</ejb-name>
  <jndi-name>StoreAccessBean</jndi-name>
  <local-jndi-name>StoreAccessLocal</local-jndi-name>
  <ejb-ref>
    <ejb-ref-name>ejb/CustomerLocal</ejb-ref-name>
    <jndi-name>CustomerLocal</jndi-name>
  </ejb-ref>
  <ejb-ref>
    <ejb-ref-name>ejb/ManagerLocal</ejb-ref-name>
    <jndi-name>ManagerLocal</jndi-name>
  </ejb-ref>
  <ejb-ref>
    <ejb-ref-name>ejb/ItemLocal</ejb-ref-name>
    <jndi-name>ItemLocal</jndi-name>
  </ejb-ref>
  <ejb-ref>
    <ejb-ref-name>ejb/SupplierLocal</ejb-ref-name>
    <jndi-name>SupplierLocal</jndi-name>
  </ejb-ref>
  <resource-ref>
    <res-ref-name>jdbc/DefaultDS</res-ref-name>
    <jndi-name>java:/DefaultDS</jndi-name>
  </resource-ref>
</session>
```

Ahora el trozo resultante de los cambios es:

```
<session>
  <ejb-name>StoreAccess</ejb-name>
  <jndi-name>StoreAccessBean</jndi-name>
  <local-jndi-name>StoreAccessLocal</local-jndi-name>
  <ejb-local-ref>
    <ejb-ref-name>ejb/CustomerLocal</ejb-ref-name>
    <jndi-name>CustomerLocal</jndi-name>
  </ejb-local-ref>
  <ejb-local-ref>
    <ejb-ref-name>ejb/ManagerLocal</ejb-ref-name>
    <jndi-name>ManagerLocal</jndi-name>
  </ejb-local-ref>
  <ejb-local-ref>
    <ejb-ref-name>ejb/ItemLocal</ejb-ref-name>
    <jndi-name>ItemLocal</jndi-name>
  </ejb-local-ref>
  <ejb-local-ref>
    <ejb-ref-name>ejb/SupplierLocal</ejb-ref-name>
    <jndi-name>SupplierLocal</jndi-name>
  </ejb-local-ref>
  <resource-ref>
    <res-ref-name>jdbc/DefaultDS</res-ref-name>
    <jndi-name>java:/DefaultDS</jndi-name>
  </resource-ref>
```

```
</session>
```

Para el message driven bean del capítulo 7 del tutorial Tusc del Lomboz, el descriptor jboss.xml generado por Lomboz no solo tiene el error anterior en el bean session StoreAccess sino que también tiene un nuevo bug que se refleja porque cuando hacemos deployment JBoss muestra el siguiente log:

org.jboss.deployment.DeploymentException: Error in jboss.xml for Bean RequestItems: ejb-ref ejb/StoreAccess found in jboss.xml but not in ejb-jar.xml

Para solucionarlo donde pone en negrita ejb/StoreAccess hay que poner StoreAccess simplemente:

```
<message-driven>
<ejb-name>RequestItems</ejb-name>
<destination-jndi-name>queue/MdbQueue</destination-jndi-name>
<ejb-local-ref><===== "ESTA TAG ESTÁ RETOCADA PUES JBOSS GENERA <EJB-REF> QUE ESTÁ MAL.
<ejb-ref-name>ejb/SupplierLocal</ejb-ref-name>
<jndi-name>SupplierLocal</jndi-name>
</ejb-local-ref><===== "ESTA TAG ESTÁ RETOCADA PUES JBOSS GENERA <EJB-REF> QUE ESTÁ MAL.

<ejb-ref>
<ejb-ref-name>ejb/StoreAccess</ejb-ref-name>
<jndi-name>StoreAccessBean</jndi-name>
</ejb-ref>
</message-driven>
```

Y así la línea negrita quedaría corregida como:

```
<ejb-ref-name>StoreAccess</ejb-ref-name>
```

Solución al problema de ejb not bound del bmp del capítulo 5 del tutorial.

Hay que sustituir un comp_name por jndi_name de una de las clases util en donde da el problema de ejb not bound. Esto habría que hacerlo cada vez que se hace EJB Generate. Pero lo mejor si queremos evitar tener que hacer esto cada vez que se hace el EJB generate, es hacer lo que pone en la página web:

http://sourceforge.net/forum/message.php?msg_id=2149132

De esta forma se generarán bien las clases util.

Solucion para que cuando incluyamos métodos ejbCreate en los ficheros del directorio src, automáticamente se incluya un método create asociado en la interfaz home.

Debemos poner un tag xdoclet a nivel de clase en el fichero correspondiente del directorio src. La tag es:

`@ejb.create-method`

Por ejemplo:

```
/**
 * @ejb.create-method
 *
 * @param alto
 * @param ancho
 * @param cerrada
 * @param idHab
 * @param luz
 * @param mascara
 * @param nombre
 * @param piso
 * @param posX
 * @param posY
 * @param puertas
 * @return
 */
public HabitacionPK ejbCreate(int alto,int ancho,boolean cerrada,String idHab,boolean
luz,String mascara,String nombre,int piso,int posX,int posY,Collection puertas)throws
javax.ejb.CreateException{
    System.out.println("Entering HabitacionBean.ejbCreate()");
    System.out.println("Leaving HabitacionBean.ejbCreate()");
    return null;
}
```

Solución al problema de que el lomboy ignora los breakpoints:

Esto se soluciona abriendo la siguiente ventana: Menú Window→Show View→ Debug

Solución para evitar errores de memoria de windows al intentar abrir un proyecto:

- Posiblemente se trate de un problema aislado, pero si al crear un nuevo Lomboz EJB Project con el nombre del proyecto que deseas abrir, recibes el ítem de error "The import javax.ejb cannot be resolved" muchas veces repetido, seguido de un mensaje de windows de allocated memory error (siempre en la misma dirección de memoria), y se te cierra la aplicación y ocurren otras cosas como no poder abrir con tu navegador servlets y cosas así, prueba a reinstalar el JDK, la misma versión o superior. En mi caso particular reinstale la misma versión (1.4.2) que obtuve de otra localización y se solucionó. Cabe destacar que al instalarlo la primera vez obtuve el error "finalizado con fallos" en dos archivos.

En el foro de la dirección de arriba viene una pequeña lista de programas que producen incompatibilidades con Lomboz, aunque no me suena ninguna.

Crea tu propia historia

Tutorial

1. Que necesitas para crear tu historia.

Con este documento pretendemos dar las indicaciones necesarias para que con poco esfuerzo y sin tener que conocer prácticamente nada de como funciona la aplicación a nivel de código, se pueda crear una historia funcional para la aplicación AGM.

La premisa básica del juego y de cada historia es obtener aprobados de maneras poco ortodoxas, pero eso no supone ninguna limitación a la hora de crear aventuras en el mundo virtual agm. Tan sólo ha de tenerse claro que los objetos son los que marcan el estado de los personajes en las historias, sin ellos no podrán realizar las acciones necesarias para obtener el objetivo fijado por la historia que se cree.

Para crear una nueva historia, hace falta lo siguiente:

- Una clase por cada objeto (no personaje o personaje no jugador) involucrado en la historia.
- Una imagen asociada a cada estado que tengan los objetos. Se ha de tener en cuenta que dicha imagen es la representación gráfica del objeto. Deben tratarse de archivos gif.
- Si la historia tiene nuevas localizaciones (habitaciones), debes tener una imagen jpg por cada una de estas nuevas habitaciones, que hará las veces de fondo.
- Modificar el GestorSistema.java del paquete Servidor.kernel de la aplicación para que se creen las nuevas habitaciones y objetos de la nueva historia al crear el mundo virtual de AGM.

2. Rango de acciones posibles.

Las acciones con las que puedes contar para dar forma a una nueva historia son:

- Hablar (Sólo con personajes no jugadores, para los personajes jugadores se usa chat. Muy útil para sacar pistas, obtener objetos, etc...).
- Mirar (Muy útil para describir los objetos y dar pistas sobre en que podrían usarse).
- Coger (Para coger objetos de una localización y usarlos después en otra).
- Usar (Con un sólo objeto, sirve para usar dicho objeto dependiendo de su funcionalidad, se producirá una acción u otra, por ejemplo, usar una lámpara serviría para encender la luz o apagarla).
- Usar (Con dos objetos, supone hacer funcionar un objeto con otro, por ejemplo, una llave con una puerta).
- Ir a (se usa para desplazarse por el mundo y sólo tiene sentido realizarla sobre lo que denominamos puertas, que son objetos especiales que comunican unas habitaciones con otras).

Los resultados que estas acciones pueden devolver son:

- Meter un objeto nuevo en el inventario del jugador.
- Sacar un objeto nuevo del inventario del jugador.
- Modificar el estado de un objeto.
- Sacar a un personaje jugador de la habitación en la que se encuentra.
- Meter a un personaje jugador en una nueva habitación.

La respuesta de una acción no está limitada a una sola opción de las mencionadas arriba, siendo posible que se produzcan un número ilimitado de ellas por acción. Por ejemplo, si cambiáramos

objetos con un personajes no jugador, la secuencia de resultados que devolvería dicha acción sería:

- Sacar ObjetoA de inventario.
- Meter ObjetoB en inventario.

3. Creando los objetos.

Objetos no personaje.

Los objetos no personaje son todos aquellos objetos inanimados que existen en el mundo de AGM. Cada uno tiene un comportamiento propio antes las acciones que los jugadores pueden realizar sobre ellos.

Para crear estos objetos de una manera sencilla, unto a este documento, se adjunta una clase “Plantilla” de la cual se pueden sacar tantas copias como objetos se deseen introducir en la historia. Después, bastará con modificar cada copia para que se conviertan en los objetos que deseamos.

Lo primero que hay que determinar, son los estados que pueden tener los objetos. Según su estado, se comportarán de una manera u otra ante las acciones que se realicen sobre ellos. Por ejemplo, creamos un objeto “caja de madera” podría tener dos estados, abierta o cerrada. Hay un límite de dos estados para cada objeto. Se representan con *strings*.

Lo primero que haremos pues es modificar el método *cambiaEstado()* de la plantilla, sustituyendo los strings ESTADO-1 y ESTADO-2 por el nombre que queramos asociar a los estados de nuestro objeto. En nuestro caso, pondríamos CERRADA y ABIERTA.

A continuación definiremos el comportamiento de nuestro objeto, con el método *ejecutarAccion(String idAcción, String idOd, String idPJ)*.

Este método es llamado por el servidor (El bean habitación donde está alojado el objeto) cada vez que los jugadores hacen una acción. La información de la acción realizada es expresada por los parámetros, *IdAccion* indica la acción, que puede ser:

- “MIRAR”
- “USAR”
- “COGER”
- “IR A”

El *idOd* indica el objeto destino sobre el que se realiza la acción, en los casos en los que el objeto en curso se está usando sobre otro (Caso: usar ObjA con ObjB, en este ejemplo, ObjB es el objeto destino). Por último el *idPJ* es el identificador del jugador que realizo la acción.

Por tanto, para cada acción para la que el objeto que queremos crear haga algo, debemos comprobar el valor de *idAccion*. En la plantilla viene como ejemplo tan solo la acción mirar, tal que así:

```
If ("MIRAR".equals(idA)) ...
```

Ahora, dentro de cada una de estas condiciones, podemos incluir otras condiciones más, por ejemplo una que compruebe el estado del objeto, y en función de dicho valor, haga una cosa u otra (por ejemplo, nuestra caja de madera, cuando la miremos y esté CERRADA, devolveremos un mensaje que diga que la caja está cerrada, y cuando hagamos lo mismo cuando esté en estado ABIERTA, devolveremos un mensaje informando del contenido de la caja).

Quedaría tal que así:

```
if (this.estado.equals("ESTADO-1")) ...
```

Otras condiciones que podemos contemplar: Comprobar que el objeto destino es el que esperábamos para realizar una acción (Imaginemos que para abrir la caja del ejemplo, necesitamos usar sobre ella una palanca, para forzarla, en ese caso, el objeto destino podría ser la palanca).

Para realizar dicha consulta, como ningún objeto se conoce entre sí, habrá que pedir a la habitación que lo busque, para ello, usaremos este código:

```

...
String nombre;
ObjetoNoPersonajeClase oNPC = null;
if(IdOd!=null){
    oNPC = h.buscarObjeto(IdOd,idPJ);
    if(oNPC!=null){
        String tipo = oNPC.getTipo();
        int ind = tipo.lastIndexOf(".");
        nombre = tipo.substring(ind+1);
    }
    else
        nombre="";
}
else
    nombre="";
if((nombre.equals("NOMBRE DEL OBJETO QUE QUEREMOS QUE INTERACTUE ON EL
QUE ESTAMOS CREANDO"))&&(estado.equals("PASTILLA"))){

```

...

La última de las comprobaciones que podemos hacer, consiste en evitar que un personaje pueda repetir acciones que no se desea que se repitan (por ejemplo, si la caja está llena de nueces, permitir coger tan sólo una nuez).

Para ello añadiremos un atributo a nuestro objeto:

LinkedList PJs;

Lista que usaremos para añadir a los personajes según vayan realizando esa acción que tan solo queremos que hagan una sola vez. Para hacer esto, usaremos el siguiente código:

```

if (!(PJs.contains(IdPj))){
    PJs.add(IdPj);
    guardarLista();
}
...

```

Ahora es cuando llega el momento de definir el comportamiento del objeto. Cuando se dan las condiciones que hemos mencionado, creamos una instancia de la clase Acciones redefiniendo sus métodos más importantes, *ejecutar(String idHab,String idPJ)* e *interpretaCondición (Habitacion hab1,Habitacion hab2,String idPJ)*. Este segundo método, aunque puede dar muchos grados de libertad al juego, es bastante complicado y detallarlo rompería con el esquema básico de este tutorial de ser una cosa sencilla y de no entrar en demasiados detalles de implementación.

En cambio en el primero, basta con meter uno de los siguientes fragmentos de código, según corresponda:

- Para mostrar un texto por pantalla (ejemplo: La caja está cerrada).

Habitacion h = conseguirHabitacion(idHab);


```
ObjetoActualizacion oA = generaNotificacion(idPJ,"TEXTO A MOSTRAR");  
mostrarResultado(h,oA);
```

- Para meter un objeto en el inventario del jugador (ejemplo: Coger Caja)

```
Habitacion h = conseguirHabitacion(idHab);  
meterEnInventario(h,idPJ,"agm.objetos.objetosNoPersonaje.NOMBRE_CLASE_OBJETO",  
"TEXTO A MOSTRAR, PUEDE SER VACIO");
```

- Para sacar un objeto del inventario del jugador:

```
Habitacion h = conseguirHabitacion(idHab);  
sacarDelInventario(h,(String)parametros.get(0),idPJ,"TEXTO A MOSTRAR, PUEDE SER  
VACIO");
```

- Para cambiarle el estado al objeto (Por ejemplo, la acción "Usar Caja" serviría para abrirla, y cambiaríamos su estado a ABIERTA):

```
Habitacion h = conseguirHabitacion(idHab);  
cambiarEstadoObjeto(h,(String)parametros.get(0),idPJ,"TEXTO A MOSTRAR POR  
PANTALLA ");
```

- Para cambiar al personaje de habitación (debemos conocer el Id de la habitación destino) (En nuestro ejemplo, supongamos que la caja abierta lleva a otra dimensión, con la acción "Ir a Caja" cambiaríamos de habitación):

```
Habitacion h = conseguirHabitacion(idHab);  
Habitacion hDest = conseguirHabitacion("ID HABITACION DESTINO");  
cambioHabitacion(h,hDest,idPJ);  
ObjetosRefresco oR = generaRefresco(hDest,idPJ);  
mostrarResultado(h,oR);
```

Con esto tendríamos casi terminado nuestro objeto. Ahora, unas consideraciones finales:

- Debemos cambiar el nombre de la clase y los constructores por el nombre que queremos que tenga nuestro objeto. (En nuestro caso, Caja).
- Para los objetos en los que no deseemos usar estados, pondremos ESTADO en su atributo de estado.
- Terminado el objeto, necesitamos una imagen gif por cada estado que pueda tener, que lo represente. El nombre de la imagen debe ser el mismo que la clase, seguido de un guión bajo y el nombre del estado al que representa. (En nuestro ejemplo, tendríamos Caja_CERRADA.gif y Caja_ABIERTA.gif).

Objetos Personajes No Jugadores.

Son aquellos objetos animados o que simulan ser personajes, pero que son controlados por la aplicación. Se construyen igual que los objetos no personaje, sólo que contemplan una acción más sobre ellos: HABLAR. Cuando eso ocurre, comienza una conversación entre el jugador y el objeto que es fija y con forma de árbol, pues se le ofrecen opciones al jugador para que pueda encaminar por donde quiere llevar la conversación, dentro de unos límites.

Antes de crear personajes no jugadores, deberías crear un árbol de conversación similar a los que hay en el documento de diseño de Objetos No Personaje incluido en esta documentación.

Los personajes no jugadores siempre tienen dos estados, CONVERSANDO y ESPERANDO. Cuando están CONVERSANDO con un jugador, ningún otro jugador puede hablar con ellos.

Realizaremos nuestros OPNJ igual que los anteriores (podremos por ejemplo, mirarles, hacer que el comando Usar represente Dar un objeto, etc...).

Todo PNJ puede hablar, sino no sería PNJ, por ello debemos incluir la comprobación de que el Id de la acción sea Hablar. Una vez comprobado eso, aparte de otras comprobaciones que deseemos hacer a nuestra elección, como que si es o no la primera vez que el personaje jugador habla con él o no, etc, tendremos que comprobar lo siguiente:

```
if(estado.equals("CONVERSANDO")){  
    ...
```

Tal y como se muestra en la plantillaPNJ adjunta a este documento, y que sirve para verificar que el personaje no esté conversando.

Si está libre, iremos por la rama del else:

```
else{  
    Acciones a=teRespondo("OPCION INICIAL",idPj);  
    return a;  
}
```

El método teRespondo implementa el arbol de conversación. Recibe como parámetros, la ID de la opcion que el jugador ha elegido contestar y el id del personaje.

En este método se debe de comprobar el id de la opción elegida de igual modo que se hacia en ejecutarAcción con el Id de las acciones, y en cada caso, construir dos listas a lo sumo, una con la conversación que es "fija" y otra con las opciones que se le ofrecen al jugador.

Debemos tener en cuenta, que si la rama de la conversación que nos ocupa es final, es decir, no se generan opciones porque la conversación termina, el estado del PNJ ha de cambiarse a ESPERANDO.

Podemos ver un ejemplo de lo que debe hacer teRespondo para cada opcion aqui:

```
Habitacion h = conseguirHabitacion(idHab);  
ObjetoComunicacion o=new ObjetoComunicacion();  
LinkedList conversacion=new LinkedList();  
o.setEmpiezaJugador(TRUE O FALSE);  
conversacion.add("TEXTO QUE DICE EL JUGADOR SI EmpiezaJugador=True");  
conversacion.add("TEXTO QUE RESPONDE EL PNJ");  
conversacion.add("TEXTO DEL JUGADOR");  
conversacion.add("TEXTO QUE RESPONDE EL PNJ");  
o.setConversacionFija(conversacion);  
LinkedList opciones=new LinkedList();  
opciones.add(new Frase("RESPUESTAS POSIBLES","ID DE LA OPCION"));  
o.setIdPJ(idPJ);  
o.setNombrePNJ("NOMBRE PJ");  
mostrarResultado(h,o);
```

Pondremos EmpiezaJugador a verdadero o falso, dependiendo de si la primera frase de la conversación fija la dice el jugador o el personaje no jugador.

Podemos añadir tantas frases a la conversación fija como queramos, las opciones están limitadas a cuatro.

4. Modificando el mundo.

Una vez creados los objetos de nuestra historia, tan sólo nos queda modificar el mundo del juego para introducir los nuevos objetos en la ubicación que queramos situarlos, e incluso añadir nuevas habitaciones si así lo deseamos.

Es conveniente que conozcamos como es el mundo, y como se comunican unas habitaciones con otras.

El siguiente mapa ASCII representa el mundo de la aplicación AGM:

```
      Conserjería    Despacho
      |              |
Baño --- Hall  ----- Pasillo
```

Ahora se muestra el mismo mapa pero con la Id de cada una de las habitaciones:

```
      H5    H6
      |      |
H3 --- H2 ----- H4
```

Es importante no olvidar que debemos obtener por cada nueva habitación que queramos crear una imagen en formato jpg que nos sirva como fondo de pantalla para representar dicha habitación. Ahora debemos editar la clase GestorSistema del paquete `agm.servidor.kernel`.

En el método `creaMundo()`, debajo de la invocación a `crearDespachoX()`; debemos invocar a `crearNueva_Habitación_1()` por cada nueva habitación que creemos, y a continuación crear esos métodos nuevos más abajo.

El esquema de dichos métodos es el siguiente:

```
private void crearNueva_Habitacion(){
```

```
    String idHab = "Hi"; // Siendo i un número mayor que 6
```

A continuación creamos las puertas. Las puertas son instancias de una clase muy similar a los objetos no personaje, sobre los que sólo tiene sentido la acción mirar y Ir a, y que conocen de antemano su destino cuando se ejecuta Ir a sobre ellas.

No hay que olvidarse de incluir las puertas en toda nueva habitación, para que no se conviertan en un punto muerto y no se pueda salir de ellas.

```
    String idHabDest = "ID HAB DESTINO";
```

```
    int posX = NUMEROX; // Depende de la resolución
```

```
    int posY = NUMEROY; // Depende de la resolución
```

```
    String estado = "CERRADA";
```

```
    String idO="O"+ contObjetos++;
```

```
    LinkedList puertas = new LinkedList();
```

```
    puertas.add (new Puerta(idO,idHab,idHabDest, posX, posY, estado));
```

A continuación se asignan valores a los atributos de las habitaciones. Su nombre, si hay luz o no, su tamaño (irrelevante porque luego se redimensiona, pero puede ser útil en posteriores versiones de la aplicación), y máscara, que son cuatro valores que definen una matriz de pixeles en la que pueden dibujarse los personajes jugadores.

```
    String nombre = "PASILLO";
```

```

boolean luz = true;
int ancho = 12;
int alto = 7;
String mascara = "100.200-400.400";

```

Después pasamos a crear los objetos de la habitación, tanto los no personaje como los personaje no jugador.

```

LinkedList objetos = new LinkedList();
LinkedList pnjs = new LinkedList();

posX = ENTEROX;
posY = ENTEROY;
estado = "ESPERANDO";
String idPNJ="PNJ"+ contPNJ++;
String nombreO="NOMBRE";
pnjs.add(new NOMBRECLASE(idPNJ,nombreO,estado,idHab,posX,posY));

posX = 820;
posY = 40;
estado = "ESTADO";
idO="O"+ contObjetos++;
NOMBRECLASE o=new NOMBRECLASE(idO,estado,idHab,posX,posY);
o.setPJs(new LinkedList());
objetos.add(o);

```

Y aqui creamos la habitación:

```

crearHabitacion(idHab,nombre,luz,ancho,alto,mascara,puertas,objetos,pnjs);
}

```

De igual forma, si queremos meter nuevos objetos en habitaciones ya existentes, el procedimiento es el mismo, añadiendolos en su método "crear".

Por último, mencionar que si se crean nuevas habitaciones, debemos procurar no dejarlas inaccesibles, por ello, tendremos que crear por lo menos alguna puerta en una habitacion ya existente para dar acceso a ellas.

5. Preparando la historia.

Una vez que hemos llegado a este punto, ya sólo queda reubicar nuestros nuevos objetos, imagenes, etc...

- Los objetos no personaje van en el paquete: agm.objetos. objetosNoPersonaje.
- Los objetos personaje no jugador van en el paquete: agm.objetos. personajesNoJugadores.
- Las imágenes deben ubicarse en ../src/clienteJugador/ interfazGrafica/Imagenes.

Si todo ha ido bien y el resultado obtenido no tiene error de compilación, la historia estaría lista para jugarse.

Bibliografía:

- “Professional EJB” Rahim Adatia, Faiz Arni, Craig A. Berry, Kyle Gabhart, John Griffin. Work Press Ltd
- Tusc tutorial: <http://www.tusc.com.au/tutorial/html/>
- TUTORIAL DE LOMBOZ: <http://www.objectlearn.com/support/docs/index.jsp>
- XDOCLET DOCUMENTATION: <http://xdoclet.sourceforge.net/olddocs/>
- FORO: www.jboss.org → pestaña forums → Beginners Corner → Tutorial on J2EE using JBOSS, Eclipse and Lomboz.
(<http://www.jboss.org/index.html?module=bb&op=viewtopic&t=36910>)
- FORO: <http://forums.cookie nest.com/viewtopic.php?t=2473&highlight=>
- FORO: http://forge.objectweb.org/forum/forum.php?forum_id=360
- FORO: <http://forge.objectweb.org/projects/lomboz>

Palabras clave

Aventura Gráfica multijugador

J2EE

J2SE

EJB

Bean

Java

SQL

HSQLDB

Autorización

Autorizamos a la Universidad Complutense a difundir y a utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Jorge Koronis Pérez

Sergio Ordoño Marín

Elsa Yáñez Morante