Control Remoto de Grúa Torre

Miguel Angel González Hernández

Proyecto Integrado Ciclo Superior DAM

Departamento de Informática – I.E.S. Julio Verne

Marzo de 2017 – Junio 2017

Índice

Lista de Cambios
1.- Introducción
2.- Estudio de Viabilidad
- 2.1.- Alcance del proyecto
- 2.2.- Descripción del Sistema Actual
- 2.3.- Descripción del Sistema Nuevo
- 2.4.- Identificación de Requisitos del Sistema
  - 2.4.1.- Requisitos de información
  - 2.4.2.- Requisitos funcionales.
  - 2.4.3.- Otros Requisitos
- 2.5.- Descripción de la solución
- 2.6.- Planificación del proyecto
  - 2.6.1.- Equipo de trabajo
  - 2.6.2.- Planificación temporal
- 2.7.- Estudio del coste del proyecto

3.- Análisis del Sistema de Información
- 3.1.- Identificación del entorno tecnológico
- 3.2.- Modelado de datos
  - 3.2.1.- Modelo Entidad-Relación
  - 3.2.2.- Esquema de la base de datos
  - 3.2.3.- Datos de prueba
- 3.3.- Identificación de los usuarios participantes y finales
- 3.4.- Identificación de subsistemas de análisis
- 3.5.- Establecimiento de requisitos
- 3.6.- Análisis de clases
- 3.7.- Definición de interfaces de usuario
  - 3.7.1.- Especificación de principios generales de interfaz
  - 3.7.2.- Especificación de formatos individuales de la interfaz de pantalla
  - 3.7.3.- Identificación de perfiles de usuario

3.7.4.- Especificacion de la navegabilidad entre pantallas

4.- Construcción del Sistema

5.- Glosario de términos

6.- Bibliografía

7.- Apéndices

Lista de Cambios

- 12/05/2017 Se decidió poder crear y eliminar usuarios en la aplicación servidor.
- 17/05/2017 Se modificaron los datos que se guardaban en la aplicación android. Se decidió poder guardar un nombre para cada conexión.
- 20/05/2017 Se decidió utilizar cambiar el entorno de desarrollo de Eclipse a jMonkey Engine 3 para el desarrollo de toda la aplicación servidor, ya que facilitaba toda la parte de gráficos en 3D.
- 25/05/2017 Se restableció el diagrama de clases.

1.- Introducción

El sistema consiste en un conjunto de dos aplicaciones, una aplicación android y una aplicación de escritorio que actuara como servidor.

La aplicación Android se conectará al servidor. Este servidor simula una grúa torre, y desde la aplicación cliente poder controlar todos los movimientos de la grúa y saber el estado exacto de esta en todo momento. La aplicación móvil constará de los controles necesarios para controlar la grúa, saber el estado de la misma y conectar con el servidor deseado. El servidor mostrará en todo momento el estado de la grúa gráficamente.

El lenguaje de programación que se va a utilizar tanto en la aplicación android como la aplicación servidor será java. La conexión entre el cliente y el servidor se hará mediante el protocolo TCP.

Para la parte gráfica se utilizara jMonkey Engine 3, este es un motor de videojuegos libre orientado al desarrollo de videojuegos en tres dimensiones. jMonkey Engine 3 es uno de los motores 3D mas completos programados en java. jMonkey Engine es una API basada en árbol de nodos. jMonkey Engine es completamente de código abierto, y se publica bajo la licencia Berkeley Software Distribution.

2.- Estudio de Viabilidad

2.1.- Alcance del proyecto

En el proyecto se pretende crear una aplicación para Android, en la cual se puedan controlar todos los movimientos de una grúa torre. La grúa de la que hablamos seria del tipo fija, este tipo de grúa se encuentra anclada al suelo, por lo tanto no dispone en su funcionamiento maniobras de traslación. Los movimientos posibles de esta grúa serian la rotación sobre si misma, el desplazamiento del carro hacia delante o atrás, y subir o bajar el gancho.

Las grúas torre se suelen controlar son:

- Mediante una cabina en la propia torre en la cual se encuentran los controles, este es el método es menos utilizado en la actualidad.
- Mediante un control remoto el cual el operario de grúa se puede desplazar por toda la obra. Este control es voluminoso y pesado.

Cualquiera de estos dos controles permite girar la grúa hacia derecha o izquierda, mover el carro hacia delante o atrás, y subir o bajar el gancho.

Estos dos sistemas tienen sus desventajas. Gracias a nuestro sistema el transporte del control seria mucho mas sencillo y cómodo, ya que es el propio smartphone del usuario el que tiene el control.

2.2.- Descripción del Sistema Actual

La Grúa Torre es un tipo de grúa de estructura metálica, desmontable y alimentada por corriente eléctrica especialmente diseñada para trabajar como herramienta en la construcción. Esta máquina se utiliza para la elevación de cargas, por medio de un gancho suspendido de un cable, en un radio de varios metros, a todos los niveles y en todas direcciones. La grúa torre se compone de los siguientes elementos:

- Base.
- Lastre estabilizador.
- Mástil.
- Corona de giro u orientación.
- Plataforma giratoria.
- Torreta, cuspide porta flechas.
- Contra pluma o contra flecha.

- Contrapeso aéreo.
- Pluma o flecha.
- Carro de pluma.
- Polipasto o gancho.
- Tirantes de pluma y/o contra flecha.
- Cables de trabajo.
- Panel de conexiones.

A pesar de tantos elementos se puede dividir en tres partes mas generales. Como son la torre metálica, con un brazo horizontal giratorio, y los motores de orientación, elevación y distribución o traslación de la carga.

En la actualidad, las grúas torres se suelen controlar mediante un control el cual se encuentra en una cabina, esta situada en la propia torre o mediante un control remoto. Sea cual sea el método por el que manejemos la grúa, los movimientos posibles de esta siguen siendo los mismos.

Los movimientos posibles son:

- **Rotar la grúa:** la grúa rota sobre si misma hacia derecha o izquierda.
- **Mover el carro:** trasladar el carro hacia delante o atrás.
- **Mover el gancho:** Subir o bajar el gancho.

2.3.- Descripción del Sistema Nuevo

El sistema que se pretende desarrollar, constara de una aplicación móvil para Android con la cual podremos conectarnos a un servidor y una vez conectados, podremos controlar todas las funcionalidades de la grúa torre. Para conectarnos necesitaremos que el servidor y el cliente

Android estén en la misma red local, o también podemos conectarnos mediante una ip publica o dominio. También necesitamos que estos dos estén iniciados.

La aplicación Android constara de todos los controles para manejar la grúa, ademas almacenará la dirección ip o dominio de la grúa, para conectarse en futuras ocasiones. Esta podrá almacenar varias grúas, las cuales aparecerán en un listado.

Respecto al servidor, este sera el encargado de controlar la grúa según las ordenes que le mande la aplicación Android.

Este sistema permitirá que el operario de grúa tenga una mayor movilidad por la obra, dado que este no tendrá que cargar con el control que se utiliza en la actualidad.

2.4.- Identificación de Requisitos del Sistema

2.4.1.- Requisitos de información

IR - 01. Conexión	
Descripción	La aplicación Android deberá guardar la información referente a la conexión con el servidor.
Datos específicos	• Nombre de la conexión • Dirección ip o dominio • usuario • contraseña
Volumen de información	Se almacenaran tantas conexiones como el usuario necesite.
Observaciones	Estos datos son los requisitos para la conexión de la aplicación Android con el servidor. Esta información la guardara la aplicación cliente.

IR - 02. Datos de sesión	
Descripción	El servidor guardara un listado de usuarios con sus contraseñas.
Datos específicos	• usuario • contraseña
Volumen de información	Se almacenaran tantos usuarios y contraseñas como usuarios se requieran.
Observaciones	Estos datos son los requisitos para la autentificación de la conexión entre la aplicación Android con el servidor.

2.4.2.- Requisitos funcionales

FR - 01. Creación de nueva conexión a grúa	
Descripción	El sistema permite al usuario registrar una nueva conexión con una grúa introduciendo los datos necesarios en la aplicación Android.

FR - 02. Crear de datos de sesión	
Descripción	El sistema permite al usuario en el servidor dar de alta nuevos usuarios con su contraseña correspondiente.

FR - 03. Eliminar conexión a grúa	
Descripción	El sistema permite al usuario eliminar una conexión existente con una grúa desde la aplicación Android.

FR - 04. Eliminar datos de sesión	
Descripción	El sistema permite al usuario en el servidor eliminar usuarios con su contraseña correspondiente, para impedir el login.

FR - 05. Conexión	
Descripción	El sistema permite al usuario conectarse con una grúa almacenada en la aplicación Android.

FR - 06. Desconexión	
Descripción	El sistema permite al usuario desconectarse de la grúa conectada.

FR - 07. Movimiento	
Descripción	El sistema permite al usuario realizar los movimientos de la grúa.

FR - 08. Listar grúas guardadas	
Descripción	El sistema permite listar todas las grúas guardadas en la aplicación Android.

FR - 09. Listar usuarios	
Descripción	Listar usuarios guardados en el servidor.

FR - 10. Comprobar duplicados	
Descripción	La aplicación servidor, realizara una comprobación al registrar un usuario nuevo, en caso de duplicidad no dejara crear este.

2.4.3.- Otros Requisitos

OR – 01. Almacenamiento	
Descripción	El servidor debe tener espacio suficiente como para guardar el fichero con los usuarios y contraseñas.

OR – 02. Conexión	
Descripción	El servidor y el dispositivo móvil, deben poder establecer una conexión entre los dos, ya sea mediante la misma red wifi o mediante un dominio.

OR – 03. Versión de Android	
Descripción	El dispositivo movil Android deberá disponer de Android 4.4 o superior para poder utilizar la aplicación.

2.5.- Descripción de la solución

La solución propuesta consiste en una aplicación nativa para Android, con la cual nos podamos conectar con un servidor mediante una dirección ip o dominio. En él se emula una grúa torre y muestra su estado gráficamente. La aplicación deberá guardar los datos de las conexiones en una base de datos SQLite. Mientras que el servidor sera capaz de guardar los usuarios y contraseñas de estos en un fichero xml, para verificar la conexión.

2.6.- Planificación del proyecto

2.6.1.- Equipo de trabajo

El sistema tiene como objetivo servir de proyecto de fin de curso para el CFGS de Desarrollo de Aplicaciones Multiplataforma en el I.E.S. Julio Verne (Sevilla). Todo el proceso del sistema se llevará a cabo por el alumno Miguel Angel González Hernández.

2.6.2.- Planificación temporal

13/03/2017 a **31/03/2017**	Elaboración del anteproyecto.
30/03/2017	Entrega del anteproyecto.
03/04/2017	Comienzo de la documentación del proyecto.
03/04/2017 a **10/04/2017**	Análisis funcional y diseño de la solución.
11/04/2017 a **28/04/2017**	Diseño y creación de la base de datos y el modelo de datos.
29/04/2017 a **05/06/2017**	Implementación de la solución.
06/06/2017 a **08/06/2017**	Elaboración de pruebas y corrección de errores.

2.7.- Estudio del coste del proyecto

Para la estimación del coste del proyecto se emplea la técnica LDC o “Líneas de código” basada en el tamaño del producto. Esta técnica trata de definir el tiempo y el costo del proyecto en base a la cantidad de líneas de código, el costo por línea y la tasa de desarrollo de líneas de código en función del tiempo. Para la estimación es imprescindible hacer una descomposición funcional del proyecto.

El valor esperado de líneas de código (E) se obtiene como una media ponderada de las estimaciones óptima (A), más probable (M) y pesimista (B).

$$E = \frac{A + 4M + B}{6}$$

Función	Optimista (A)	Mas probable (M)	Pesimista (B)	Esperado (E)	Précio por línea	Coste
Gestión de datos de conexión (Servidor)	150	200	350	216,67	0,50 €	108,33 €
Gestión de conexiones (Servidor)	150	200	350	216,67	0,50 €	108,33 €
Gestión de conexiones (Aplicación Android)	200	280	400	286,67	0,50 €	143,33 €
Interfaz de usuario	900	1500	2000	1483,33	0,50 €	741,67 €
Modelo de datos y base de datos	150	200	300	208,33	0,50 €	104,17 €
Control de grúa (Aplicación Android)	500	600	800	616,67	0,50 €	308,33 €
					Total:	**1.514,17 €**

3.- Análisis del Sistema de Información

3.1.- Identificación del entorno tecnológico

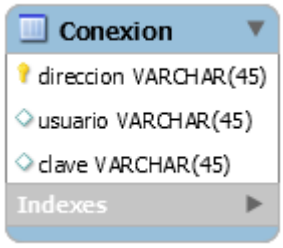
A continuación se darán a conocer todos los detalles de la infraestructura técnica que se utilizará para el desarrollo del sistema.

- Hardware:
  - PC:
    - Procesador Intel Core i5 4670k 3.4GHz
    - HDD 1TB
    - RAM 16 GB DDR3
    - Tarjeta gráfica Nvidia GTX 760

- Software:
  - Sistema operativo Windows 10 Pro 64 bits.
  - IDE Android Studio 3.3
  - Blender 2.78c
  - jMonkey Engine SDK 3.1.0 Stable

3.2.- Modelado de datos

3.2.1.- Modelo Entidad-Relación



3.2.2.- Esquema de la base de datos

La base de datos se generara de forma automática la primera vez que se ejecute la aplicación en el smartphone. Esta base de datos se almacena en un fichero en el sistema de almacenamiento del smartphone.

La aplicación utiliza SQLite como sistema gestor de base de datos. SQLite es un pequeño gestor de base de datos relacional y transaccional que funciona directamente dentro de la aplicación, no se tiene que acceder a ningún servidor.

El conjunto de herramientas de desarrollo Android SDK provee una serie de clases para administrar una base de datos SQLite. La comunicación con la base de datos se hace mediante la clase abstracta SQLiteOpenHelper. La clase encargada de la creación de la base de datos hereda de SQLiteOpenHelper obligando a implementar los métodos onCreate() y onUpdate().

```
1  CREATE TABLE conexion(
2          direccion TEXT,
3          usuario TEXT,
4          clave TEXT
5  );
```

El servidor guardara los usuarios y sus contraseñas en un fichero xml. La aplicación comprobara si existe el fichero y en caso de que no exista creara uno. Si este existente, se leerán todos los datos y una vez se valla a cerrar la aplicación esta guardara los cambios realizados.

```
1  <grúa>
2     <usuarios>
3        <usuario clave="contraseña" usuario="nombre de usuario"/>
4        <usuario clave="contraseña" usuario="nombre de usuario"/>
5        ...
6     </usuarios>
7  </grúa>
```

3.2.3.- Datos de prueba

Base de datos SQLite:

```
INSERT INTO conexion VALUES('localhost','usuario','usuario');

INSERT INTO conexión VALUES('192.168.1.2','usuario','usuario');
```

Fichero xml:

```
<grúa>
  <usuarios>
    <usuario clave="usuario" usuario="usuario"/>
    <usuario clave="contraseña" usuario="Miguel Angel"/>
    <usuario clave="admin" usuario="admin"/>
  </usuarios>
</grúa>
```

3.3.- Identificación de los usuarios participantes y finales

La aplicación Android deberá utilizarse por un operador de grúa. Dicho operador deberá tener el permiso pertinente dependiendo de las leyes establecidas en el país que se encuentre. En el caso de estar en España sera necesario el **carnet de gruista u operador de grúa torre** para el manejo de esta. Por lo tanto el usuario tendría los conocimientos necesarios de:

- Entorno social y normativa básica vigente, relacionados con la grúa torre.
- Utilización y manejo de la grúa torre en la obra.
- Reparación y mantenimiento de la grúa.
- Riesgo derivado de la instalación y utilización de la grúa torre.
- Elementos auxiliares a utilizar en la carga y descarga de materiales. Forma de realizar la carga y el trasporte, atendiendo a normas de seguridad.

Solo el operario de grúa sera el encargado de manipular el servidor y la aplicación movil.

3.4.- Identificación de subsistemas de análisis

Nuestro sistema se dividirá en dos grandes subsistemas y estos se dividen en otros.

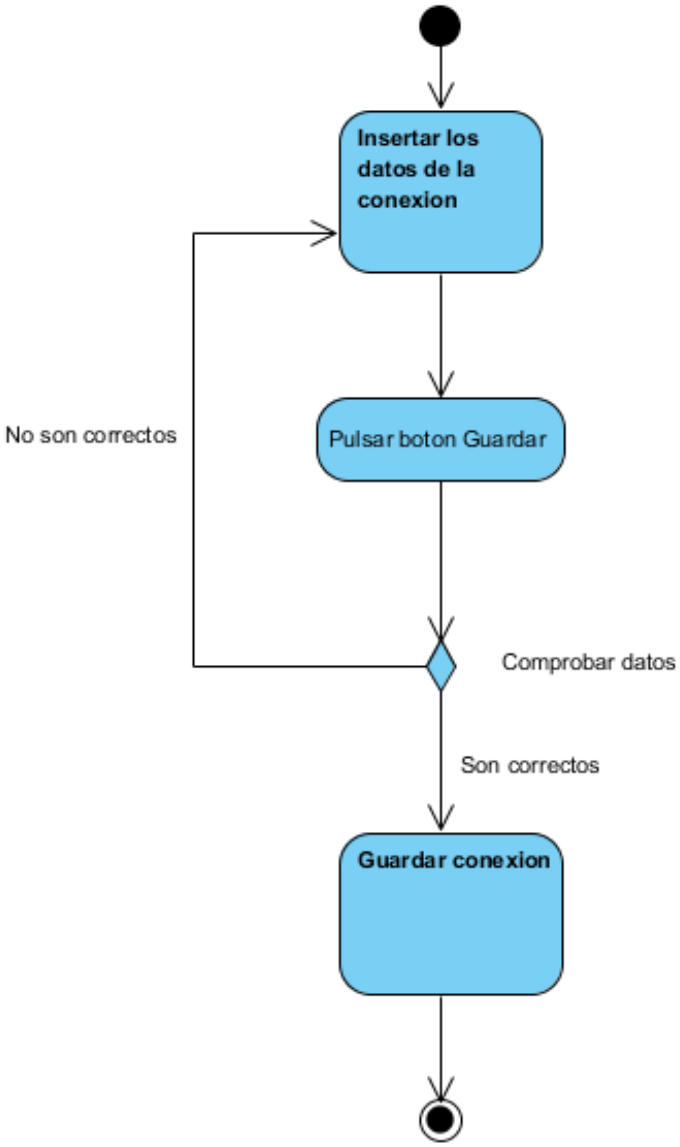
- **Aplicación Android:**
  - **Gestión de las conexiones.** Permite al usuario añadir, borrar y listar las conexiones almacenadas en la base de datos.
  - **Control de la grúa.** Permite al usuario controlar los diversos movimientos de la grúa.
  - **Conexión con el servidor.** Permite al usuario establecer la conexión con el servidor para el futuro control de la grúa.
- **Servidor:**
  - **Gestión de usuarios.** Permite al usuario añadir, borrar y listar los usuarios almacenados en el fichero xml.
  - **Conexión.** Permite al usuario conectarse mediante la aplicación Android para el futuro control de la grúa.
  - **Representación gráfica del estado de la grúa.** Permite visualizar en todo momento el estado de la grúa.

3.5.- Establecimiento de requisitos

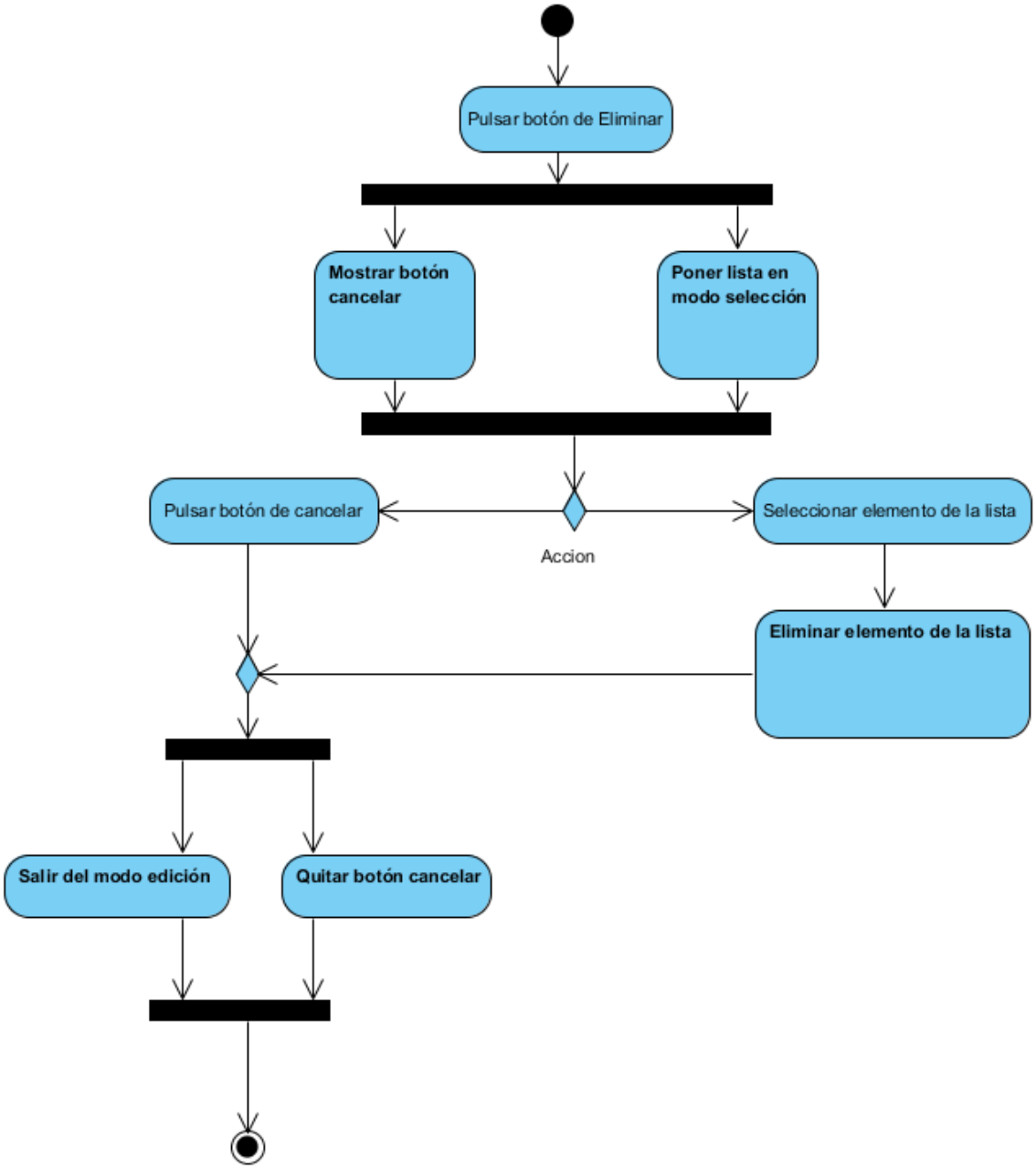
- **Aplicación Android:**
  - **Subsistema de gestión de las conexiones.**

    El subsistema provee a la aplicación de las siguientes funcionalidades para el usuario:

    - **Añadir una nueva conexión:** De este modo se pedirán por pantalla los datos para la conexión, como son: usuario, clave y dirección. Una vez introducidos estos datos la aplicación deberá comprobar que estos son correctos, en caso de ser no cumplir los requisitos se mostrará un mensaje, advirtiendo al usuario. Una vez los datos sean correctos se procederá a guardar estos datos.

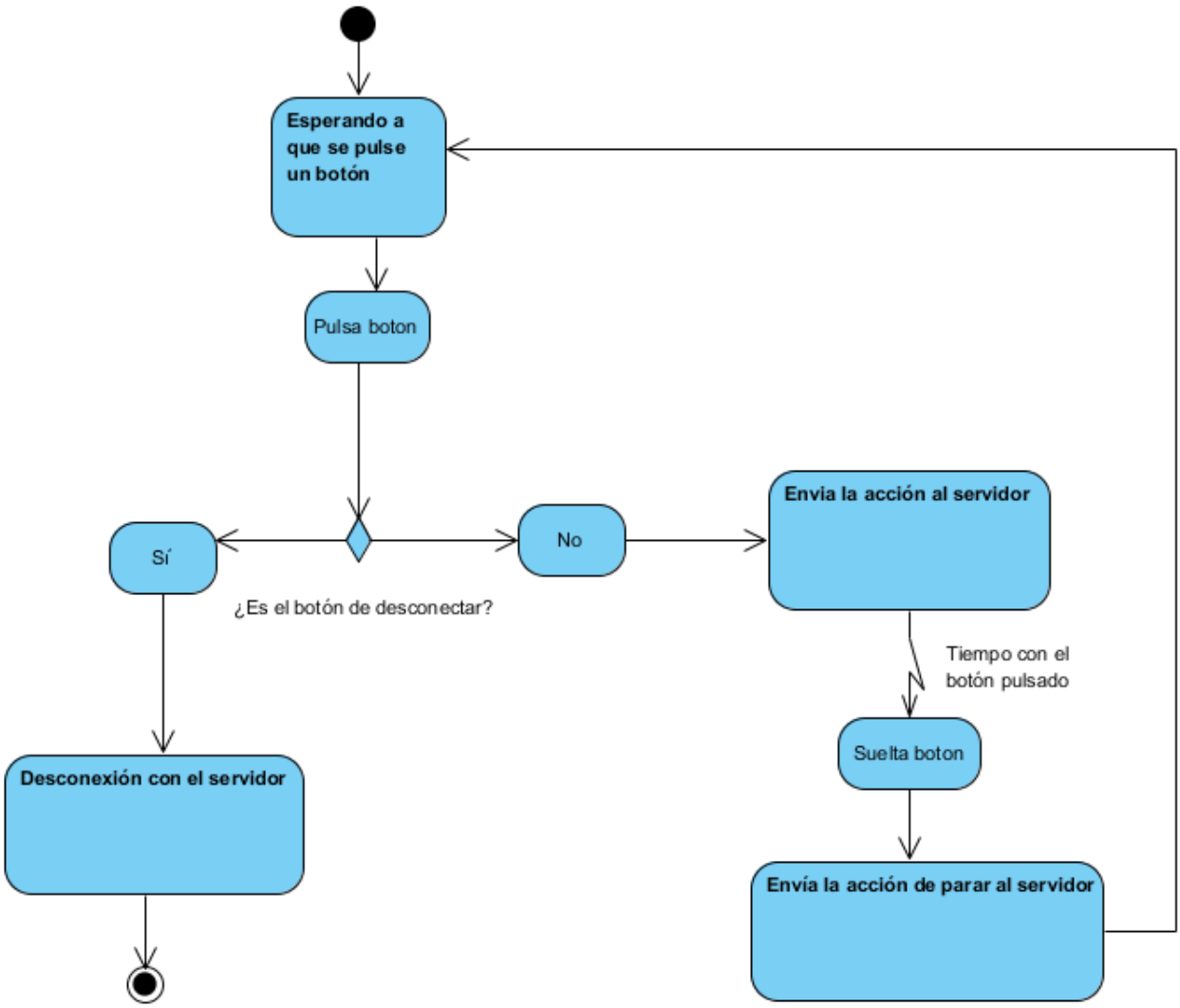


- **Borrar una conexión existente.** Una vez nos encontramos en la lista de conexiones guardadas, en el menú tendremos una opción para borrar una conexión de la lista. Bastara con seleccionar esta opción y después seleccionar el elemento de la lista que queramos borrar. En caso de no querer eliminar ningún elemento, en el menú aparecerá un botón de cancelar.

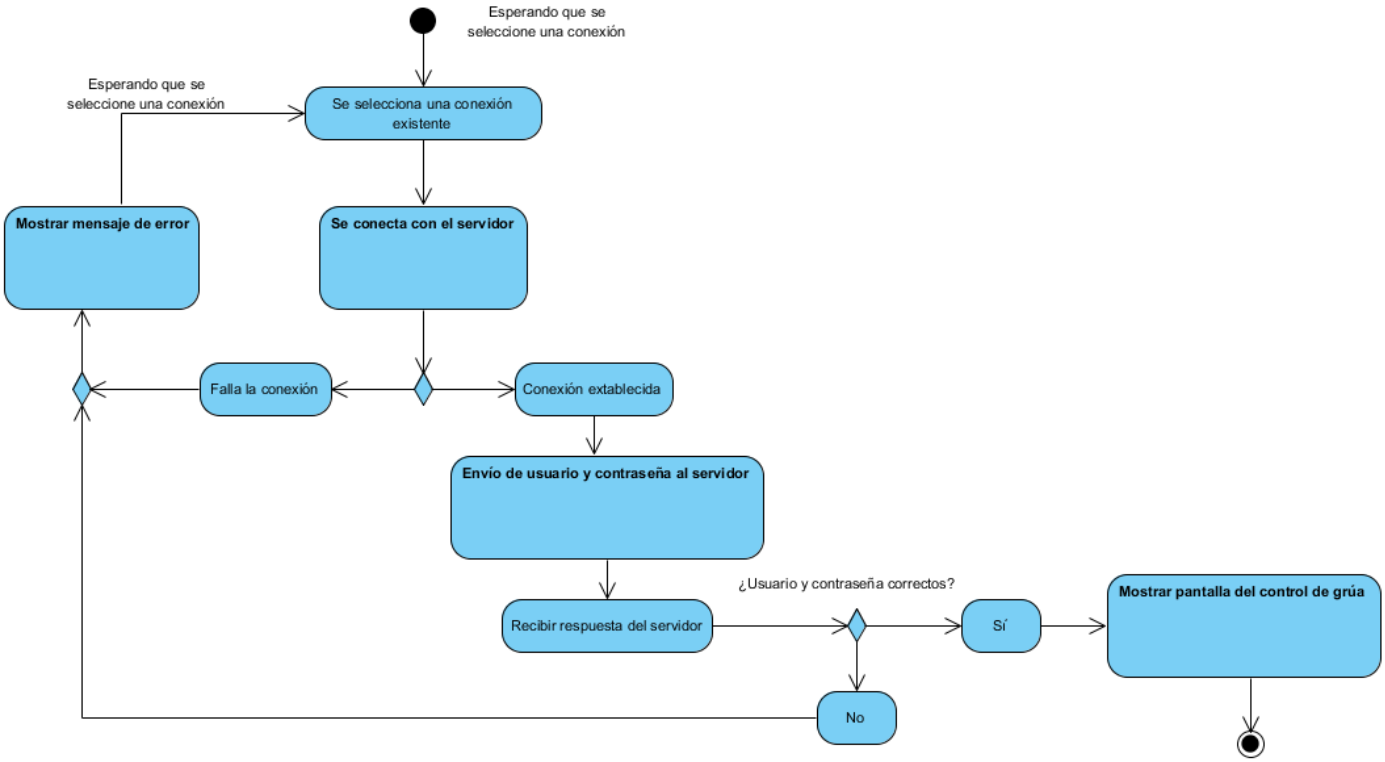


- Listar las conexiones guardadas. El sistema recogerá todos los datos de la base de datos SQLite y los mostrara en forma de lista.

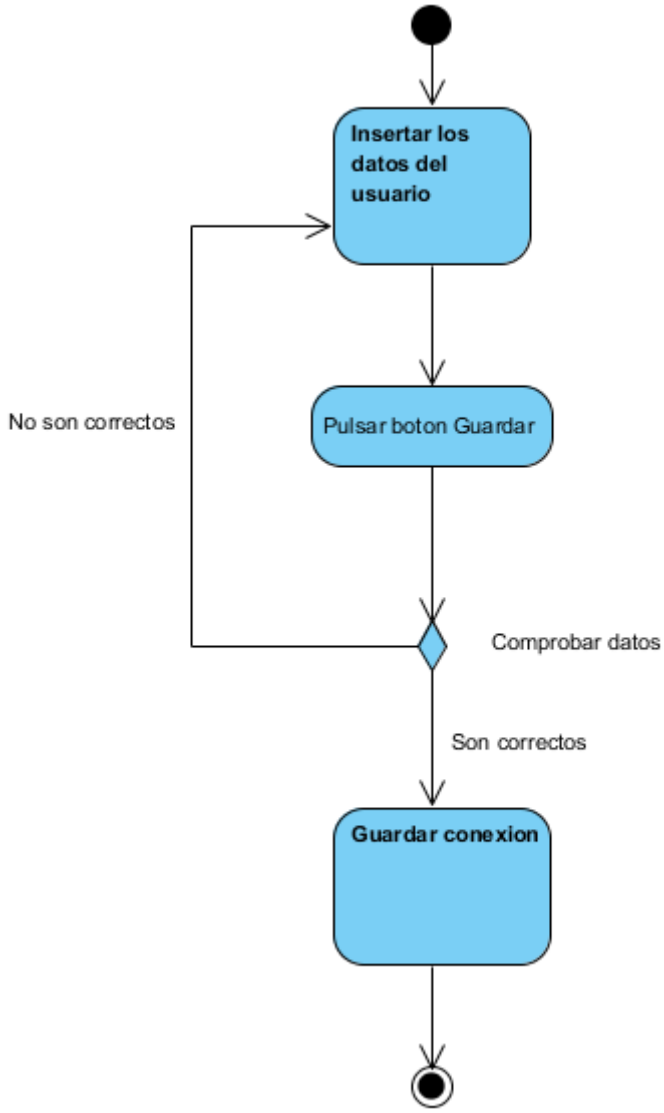
- **Subsistema de control de la grúa.** La aplicación dispondrá en la ventana de control de la grúa de todos los botones y joystick necesarios para el control de la grúa. Dependiendo de que botón se este pulsando el sistema transferirá la orden al servidor, cuando se suelte el botón mandará otra orden al servidor para que pare la acción.



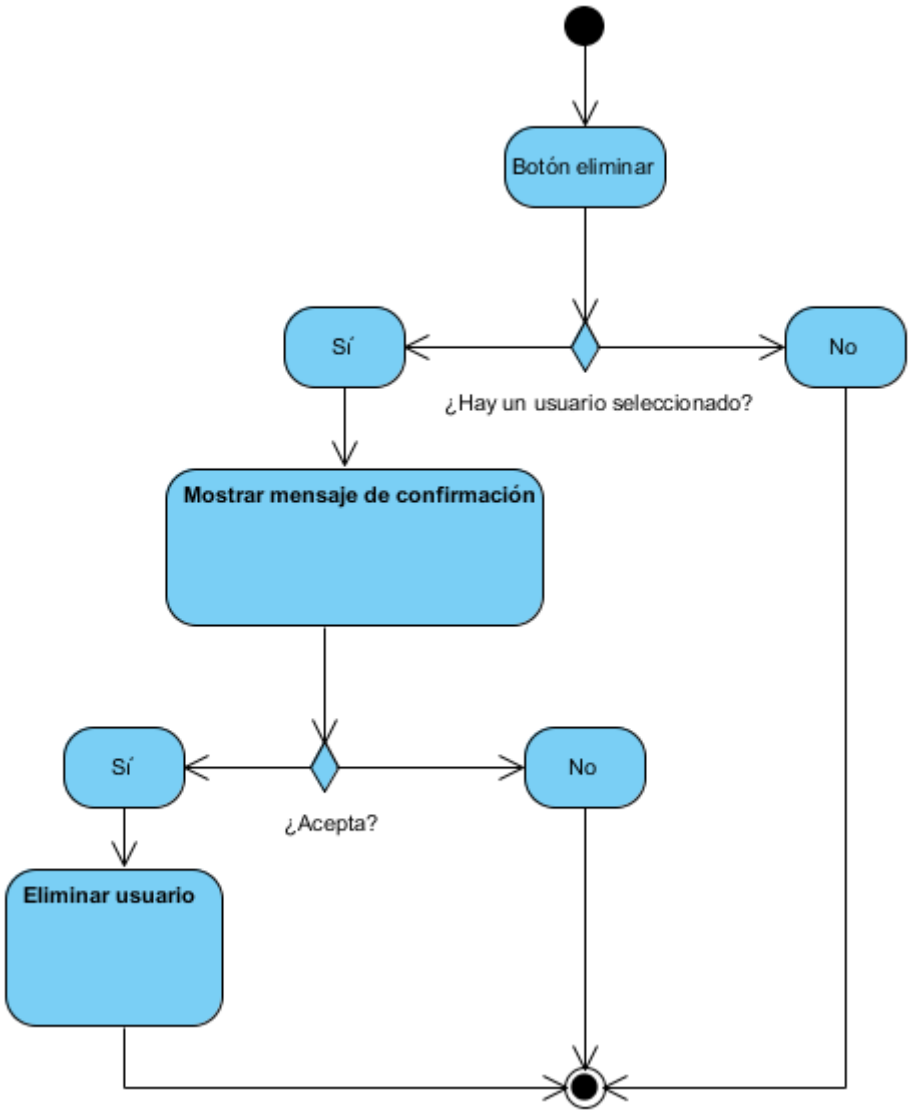
- **Subsistema de conexión con el servidor.** El sistema una vez seleccionemos una conexión en la lista de conexiones, intentara conectarse con el servidor. En caso de fallo la aplicación mostrará un mensaje de error y deberemos de seleccionar esa u otra conexión de la lista. En caso de que la conexión se establezca con éxito, la aplicación le mandará al servidor el usuario y la contraseña. El servidor comprobará esos datos y nos dará una respuesta. En caso negativo, la aplicación mostrara un mensaje de error y tendremos que seleccionar ese u otro elemento de la lista. En caso afirmativo la aplicación nos mostrara la pantalla del control de la grúa.



- **Servidor:**
  - **Subsistema de gestión de usuarios.**
    - **Guardar usuarios.** De este modo se pedirán por pantalla los datos del usuario, como son: nombre de usuario, clave. Una vez introducidos estos datos la aplicación deberá comprobar que estos son correctos, en caso de ser no cumplir los requisitos se mostrará un mensaje, advirtiendo al usuario. Una vez los datos sean correctos se procederá a guardar estos datos.



- **Borrar usuarios.** Una vez nos encontramos en la lista de usuarios guardados, seleccionaremos el usuario que queremos borrar. Una vez lo tengamos seleccionado, pulsaremos el botón de eliminar. Nos mostrara un mensaje de confirmación y si aceptamos se borrara el usuario del fichero xml, en caso contrario no haremos nada.



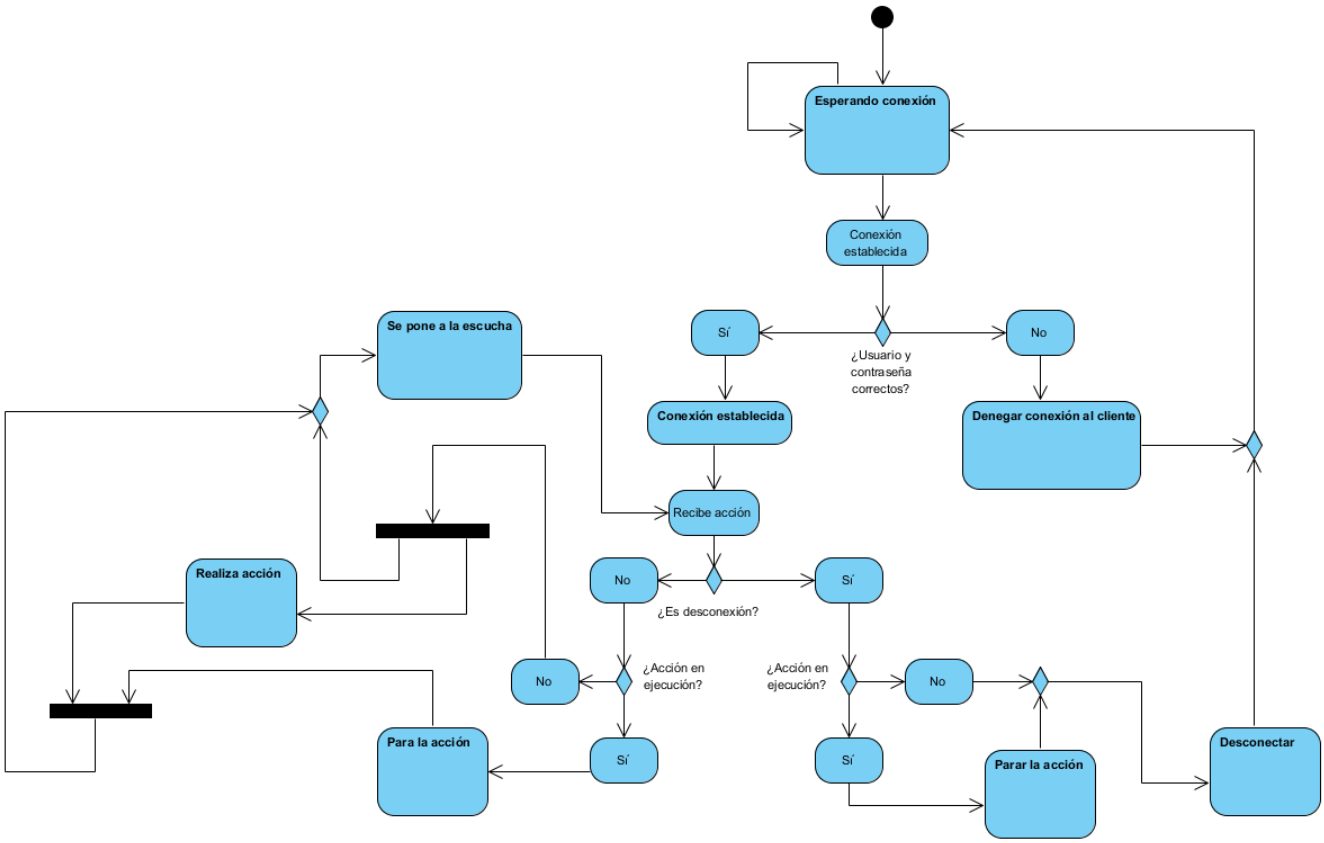
- **Listar usuarios.** El sistema recogerá todos los datos de los usuarios, del fichero xml y los mostrara en forma de lista.

- **Subsistema de conexión.** Una vez el servidor se inicia y carga todos los datos de los usuario. El servidor se pone en modo escucha, esperando a que se conecte un cliente. Cuando un cliente establece una conexión, el cliente le envía al servidor los datos del usuario(nombre de usuario y clave). Si estos datos son incorrectos el servidor se desconecta del cliente y vuelve a la escucha.

  Si los datos son correctos, el servidor espera a que el cliente le mande una acción para realizar. El servidor manda la acción a la grúa y vuelve a escuchar. Si recibe una acción

y la grúa esta haciendo una acción, este se vuelve a poner a la escucha. Cuando recibe la parada de una acción, para la acción correspondiente y vuelve a la escucha.

Si el servidor recibe la acción de desconectarse, se desconecta, pero si hay alguna acción en curso, este para la acción y posteriormente se desconecta.

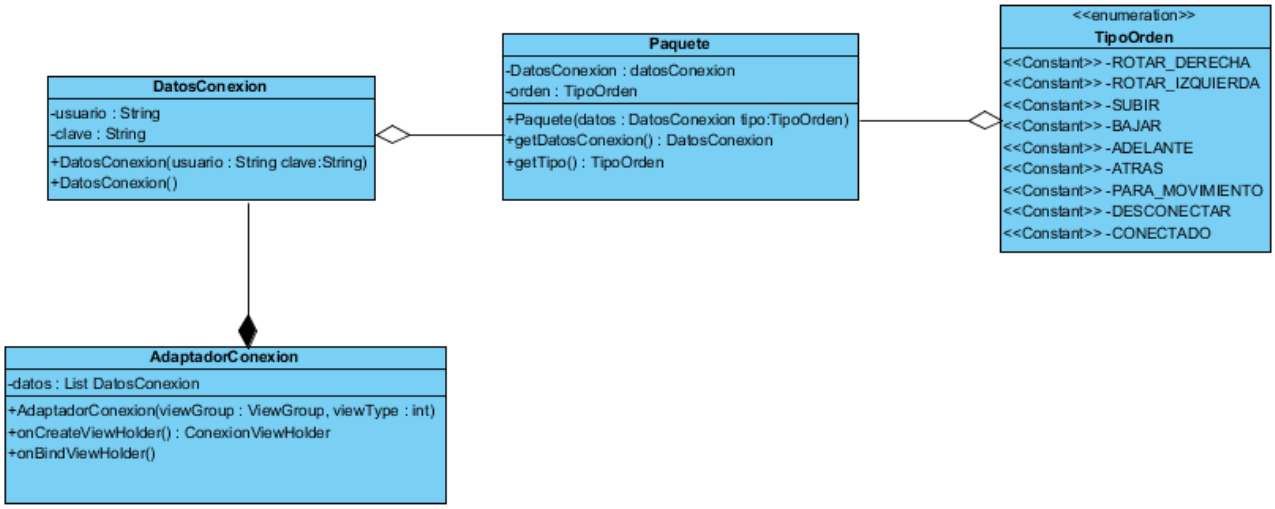


- **Subsistema de representación gráfica del estado de la grúa.** El sistema de representación gráfica, recibe las acciones enviadas por el cliente y representa gráficamente la acción por pantalla. Por ejemplo, el servidor recibe la acción de rotar a la derecha, este sistema haría que la grúa se moviese a la derecha.

3.6.- Análisis de clases

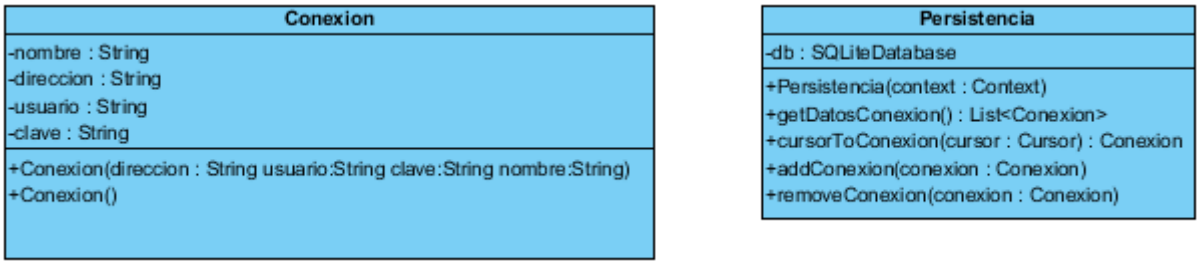
3.6.1.- Comunes

Clases comunes para el servidor y el cliente.



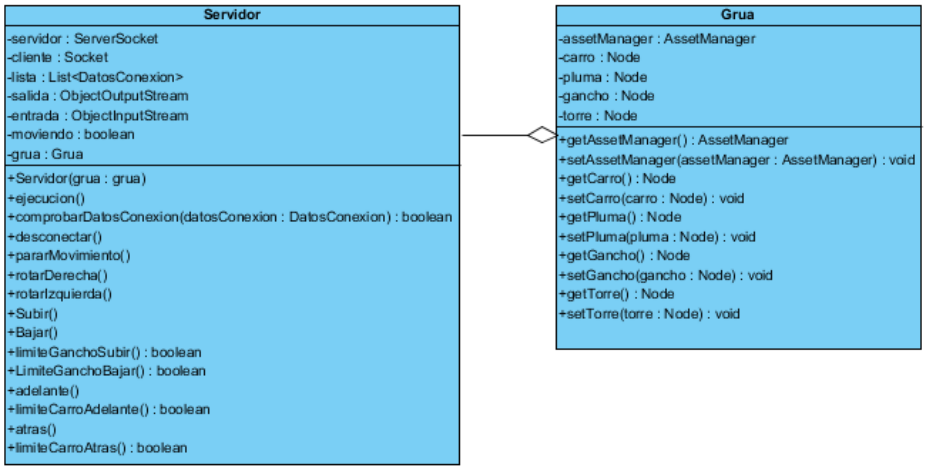
3.6.2.- Aplicación Android

Clases pertenecientes a la aplicación android.



3.6.3.- Aplicación servidor

Clases pertenecientes a la aplicación servidor.



3.7.- Definición de interfaces de usuario

3.7.1.- Especificación de principios generales de interfaz

Todas las pantallas pertenecientes a la aplicación Android, seguirán una gama de colores similar, también tendrán consistencia en las fuentes y sus tamaños.

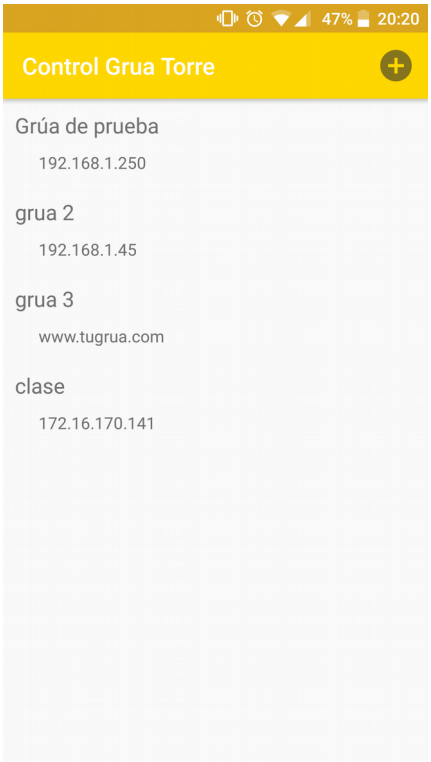
Colores:

#FFD700 → Color primario
#DAA520 → Color para la barra de notificaciones
#00008B → Color acentuado
#FFFFFF → Color de fondo

3.7.2.- Especificación de formatos individuales de la interfaz de pantalla

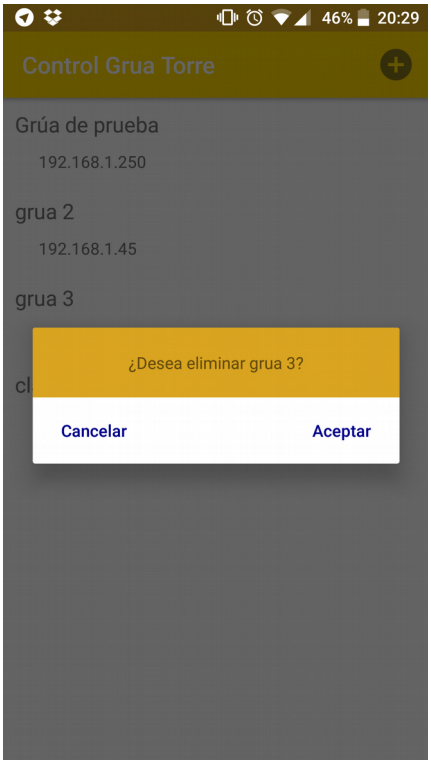
Inicio:

Una vez iniciada la aplicación, nos encontraremos en una pantalla en la cual contendrá una lista de las grúas que tenemos almacenadas. En esta tenemos la opción de conectarnos a una de estas haciendo clic, borrarla haciendo clic continuo o crear una nueva.



Eliminar conexión:

En el caso de mantener pulsado una grúa, la aplicación nos preguntara si deseamos eliminar la conexión a la grúa.

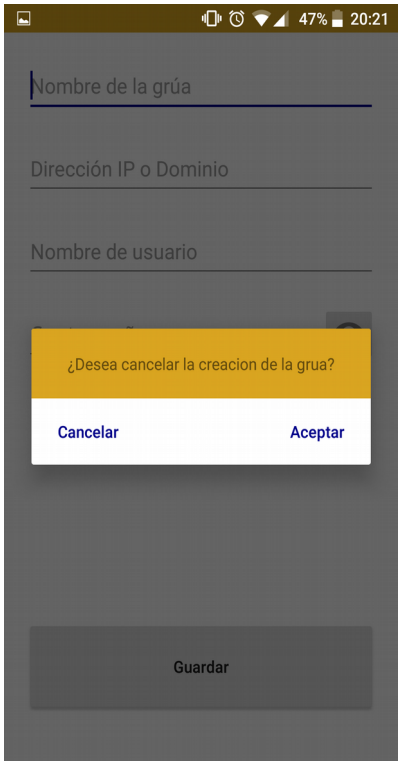


Crear conexión:

En el caso de añadir una nueva grúa, pulsaremos el botón de mas situado en la barra de menú a la derecha. Al hacer esto pasaremos a una pantalla en la cual podremos insertar todos los datos de nuestra nueva grúa para guardarla posteriormente.

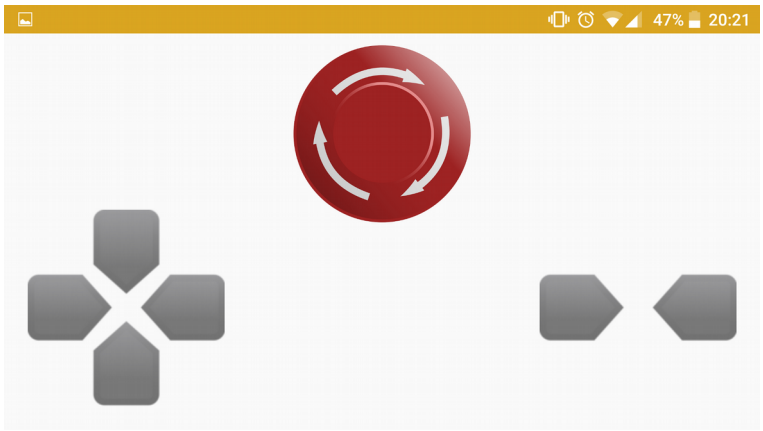
Cancelación de la nueva grúa:

En el caso de querer cancelar el proceso de añadir una grúa, al pulsar el botón de atrás nos aparecerá un mensaje en el cual deberemos aceptar.



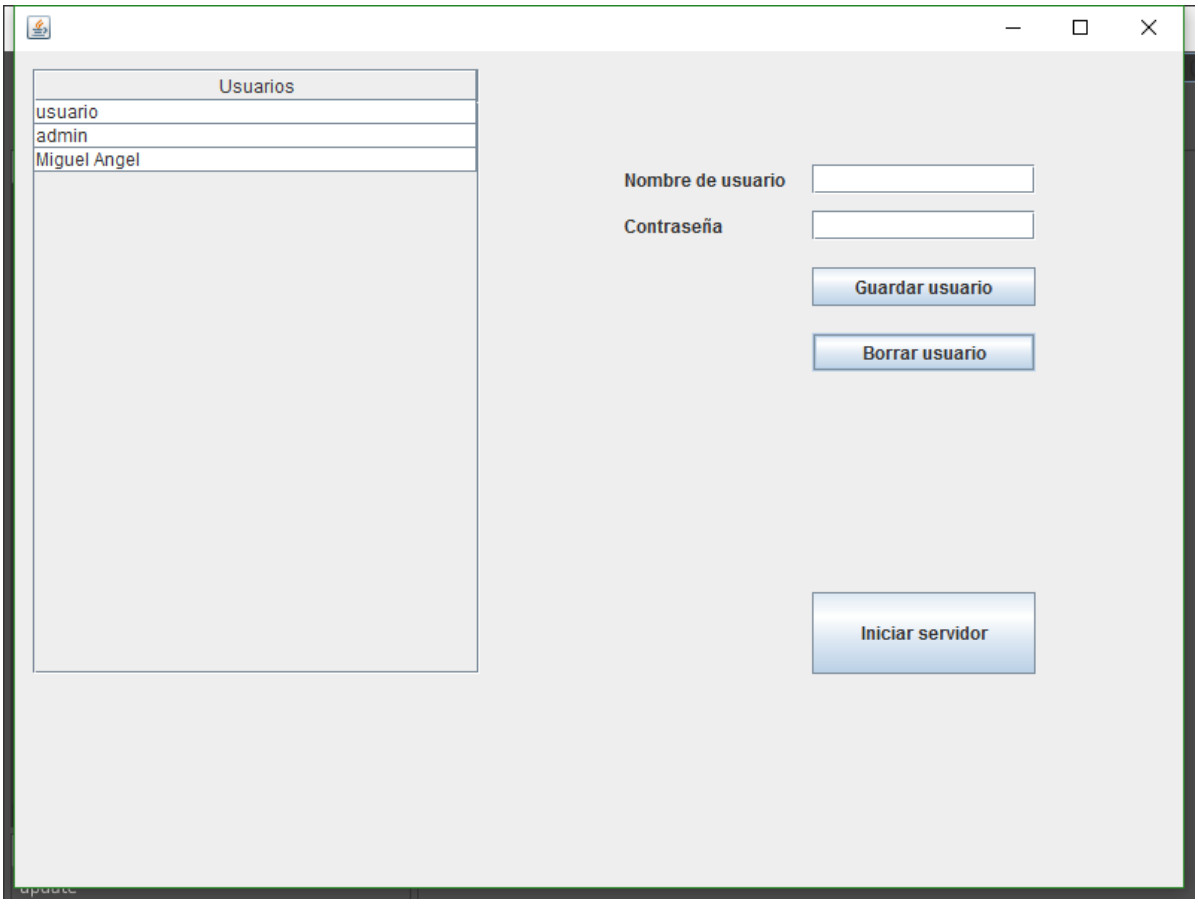
Control:

En el caso de elegir una grúa, nos conectaremos a ella y para poder controlarla nos aparecerá el siguiente control. Aquí tenemos de izquierda a derecha los controles para controlar la rotación de la grúa y la elevación del gancho, el botón de parada de emergencia y por ultimo los botones para controlar la posición del carro.



Pantalla del servidor:

Esta pantalla nos permite crear usuarios, borrarlos y ver todos los usuarios que existen en el servidor. Cambien si pulsamos en el botón de iniciar servidor, se ejecutará el servicio de escucha y la grúa en 3D.



Pantalla de configuración 3D:

En esta pantalla podemos elegir configuraciones como son si lo queremos en pantalla completa o no, la resolución, sincronización vertical, corrección de color, anti-Aliasing, y la velocidad de refresco.

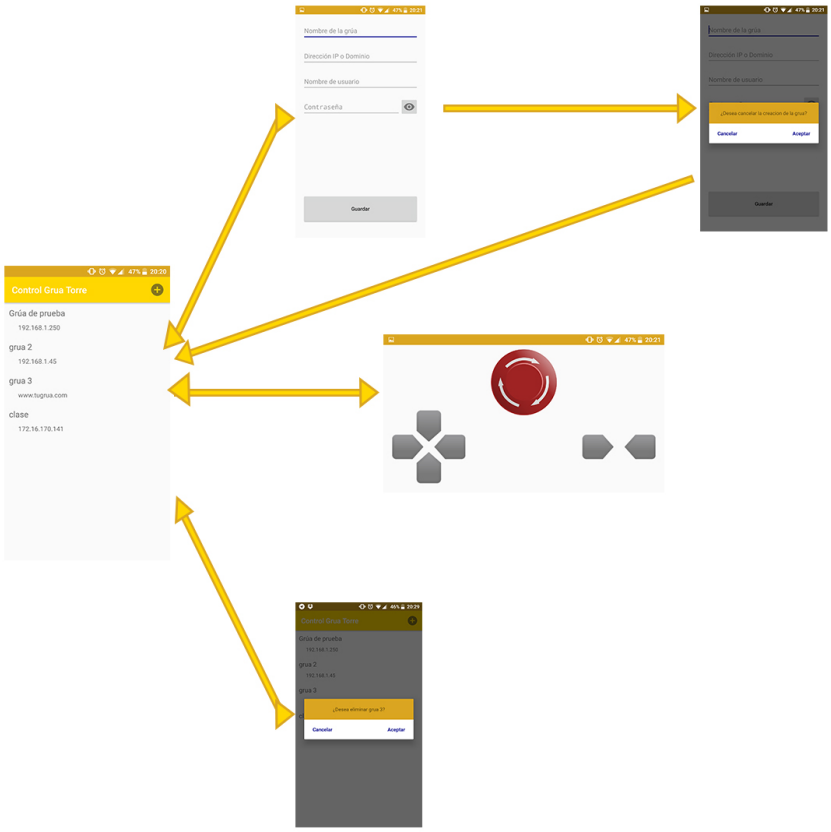
Visualización de grúa en 3D:

Una vez tenemos elegida la configuración le damos a continue y nos cargara una pantalla similar a esta, dependiendo de la configuración que hayamos elegido.

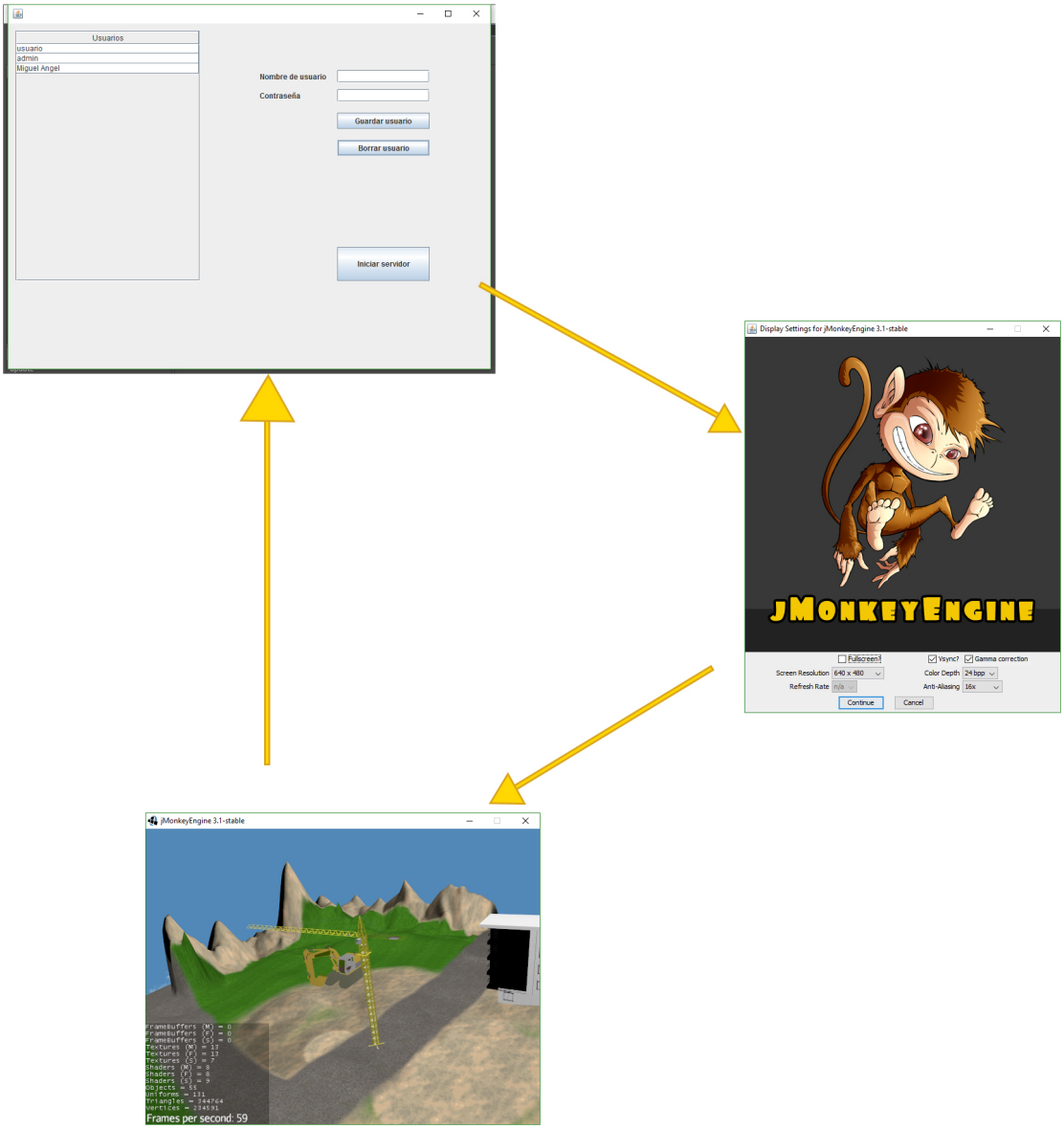
3.7.3.- Especificación de la navegabilidad entre pantallas

En la aplicación android, desde la pantalla de inicio el usuario podrá ir a crear una nueva conexión, ir a controlar una grúa o eliminar una grúa.

3.7.4.- Especificación de la navegabilidad entre pantallas



En el servidor desde la pantalla principal, podremos crear y borrar usuarios. También tenemos la opción de iniciar el servidor. Para pasar de la vista 3D de la grúa a la pantalla inicial, es tan fácil como pulsar la tecla escape.


Usuarios
usuario
admin
Miguel Angel
Nombre de usuario
Contraseña
Guardar usuario
Borrar usuario
Iniciar servidor
Display Settings for jMonkeyEngine 3.1-stable
jMonkeyEngine
Fullscreen?
Vsync?
Gamma correction
Screen Resolution
640 x 480
Color Depth
24 bpp
Refresh Rate
n/a
Anti-Aliasing
16x
Continue
Cancel
jMonkeyEngine 3.1-stable
Frames per second: 59

4.- Construcción del Sistema

4.1.- Conexión cliente servidor

El cliente se conecta mediante una conexión TCP con el servidor. Estos utilizan la clase Paquete para intercambiar información entre si. En esa clase paquete viaja la información necesaria para la comprobación de que es un usuario valido y la orden que se quiere ejecutar en cada caso.

Servidor

```
public void ejecucion() throws java.lang.ClassNotFoundException{
     System.out.println("Ejecutando Servidor");
     while(true){
          try {
               while(true){
                    cliente = servidor.accept();
                    Paquete paquete = null;
                    entrada =new
ObjectInputStream(cliente.getInputStream());
                    /
                    paquete = (Paquete) entrada.readObject();

     if(this.comprobarDatosConexion(paquete.getDatosConexion())){
          salida=new
ObjectOutputStream(cliente.getOutputStream());
          Paquete paqueteSalida = new
Paquete(paquete.getDatosConexion(), TipoOrden.CONECTADO);
          salida.writeObject(paqueteSalida);
          System.out.println("Recibiendo orden ->
"+paquete.getTipo().toString());
          switch(paquete.getTipo()){
               case ROTAR_DERECHA:
                    rotarDerecha();
                    break;
               case ROTAR_IZQUIERDA:
                    rotarIzquierda();
                    break;
               case SUBIR:
                    subir();
                    break;
               case BAJAR:
                    bajar();
                    break;
               case ADELANTE:
```

```
                    adelante();
                    break;
                case ATRAS:
                    atras();
                    break;
                case PARAR_MOVIMIENTO:
                    pararMovimiento();
                    break;
                case DESCONECTAR:
                    desconectar();
                    break;
                case CONECTADO:
                    System.out.println("conectado");
                    break;
                }
            }else{
                this.desconectar();
                salida=new
ObjectOutputStream(cliente.getOutputStream());
                paquete = new Paquete(paquete.getDatosConexion(),
TipoOrden.DESCONECTAR);
                salida.writeObject(paquete);
            }
        }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Cliente

```
public void conectar() /*throws NetworkOnMainThreadException*/{
    new Thread(new Runnable() {
      public void run() {
        try {
          Socket cliente = new Socket(host, puerto);
          Log.i("CLIENTE", "Conectado");
          Paquete paquete = new Paquete(datos, TipoOrden.CONECTADO);
                                    ObjectOutputStream  salida=  new
ObjectOutputStream(cliente.getOutputStream());
          salida.writeObject(paquete);
```

```
                                    ObjectInputStream entrada =new
ObjectInputStream(cliente.getInputStream());
         paquete =(Paquete) entrada.readObject();
         if(paquete.getTipo() == TipoOrden.CONECTADO){

         }else{
           entrada.close();
           salida.close();
           cliente.close();
         }
       } catch (IOException e) {
         e.printStackTrace();
       } catch (ClassNotFoundException e) {
         e.printStackTrace();
       }
     }
   }).start();
  }

public void movimiento(final TipoOrden orden) /*throws
NetworkOnMainThreadException*/{
    new Thread(new Runnable() {
        public void run() {
            try {
                ObjectOutputStream salida;
                ObjectInputStream entrada;
                Paquete paquete;
                Socket cliente = new Socket(host, puerto);
                Log.i("CLIENTE", cliente.toString());

                paquete = new Paquete(datos, orden);
                                                salida = new
ObjectOutputStream(cliente.getOutputStream());
                salida.writeObject(paquete);
                                                entrada = new
ObjectInputStream(cliente.getInputStream());
                paquete = (Paquete) entrada.readObject();
                if (paquete.getTipo() == TipoOrden.DESCONECTAR) {
                            System.err.println("Datos de conexion
incorrectos");
                }

                entrada.close();
                salida.close();
                cliente.close();
            } catch (IOException e) {
```

```
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }).start();
}
```

Para comprobar los datos de conexión provenientes del cliente, el servidor realiza una comprobación con todos sus usuarios cargados en memoria en el momento de la ejecución del servicio, estos usuarios provienen del fichero usuarios.xml.

Comprobar datos de conexión

```
private boolean comprobarDatosConexion(DatosConexion datos){
          boolean r=false;
          for(DatosConexion e : lista){
               if(e.equals(datos)){
                    r=true;
               }
          }
          return r;
     }
```

4.3.- Gráficos 3D

4.3.1.- Posicionar los elementos gráficos 3D

A la hora de representar la grúa de forma gráfica, se crea un mundo en el cual tenemos que añadir todos nuestros nodos. Primero debemos crear una escena, esta ha sido creado con jMonkey Engine 3, la escena es la en respecto a la que vamos a insertar los demás elementos. Una vez instanciada la escena, la añadimos a nuestro mundo. Una vez echo esto posicionamos la cámara. Instanciamos la grúa y la añadimos al mundo. También cargamos los demás elementos como son el punto de luz y los demás modelos 3D.

Iniciar la escena

```
public void simpleInitApp() {

   //Cargar escena con el mundo
    scene = assetManager.loadModel("Scenes/mundo.j3o");
    rootNode.attachChild(scene);

    posicionarCamara();
    grua = new Grua(this.assetManager);
    rootNode.attachChild(grua);
    cargarPuntoLuz();
    crearEscavadora();
    cargarPisos();

    hiloServidor();
}
```

Para colocar la obtenemos la cámara que existe al crearse el mundo y le damos una localización y una dirección en la que va a mirar. Esta localización y dirección se insertan en forma de vectores.

Posicionar la cámara

```
private void posicionarCamara() {
          viewPort.getCamera().setLocation(new Vector3f(60.f, 70.f,
80.f));
      viewPort.getCamera().lookAtDirection(new Vector3f(-10f, -10f,
-25f), new Vector3f(0f, 0f, 0f));
}
```

A la hora de crear el punto de luz debemos definir que tipo de punto de luz necesitamos, si es direccional, punto de luz simple, etc. Para nuestro caso hemos elegido un punto de luz simple, el cual solo necesita una posición que se la daremos mediante un vector. Y posterior mente la añadimos a nuestro mundo.

Cargar y posicionar punto de luz

```
private void cargarPuntoLuz() {
    PointLight lamp = new PointLight();
    lamp.setPosition(new Vector3f(50.f, 200.f, 50.f));
    rootNode.addLight(lamp);
  }
```

Para hacer mas ameno nuestro mundo hemos insertado algunos elementos mas en nuestro mundo, para colocar estos modelos 3D, es tan sencillo como cargar el modelo 3D procedente de un fichero .obj, y asignárselo a un nodo. Para que todas las dimensiones de nuestros elementos coincidan le vamos ha dar una escala, se recomienda siempre que la escala sea inferior a 1, ya que si es mayor a 1, el calculo de procesamiento gráfico se eleva y puede producir cortes o lag en los movimientos. También le asignamos una posición, en este caso elegimos pasarle las posiciones *X*, *Y* y *Z*. Pero también podíamos haberlo hecho mediante un vector o directamente insertarlos en nuestra escena a la hora de crearla.

Cargar otros modelos 3D

```
public void cargarPisos(){
    Node piso =
(Node)assetManager.loadModel("Models/pisos/piso1/building.obj");
    piso.scale(0.15f);
    piso.move(38, 0, -45);
    piso.setShadowMode(RenderQueue.ShadowMode.CastAndReceive);
    rootNode.attachChild(piso);
  }

public void crearEscavadora(){
   Node escavadora = (Node)
assetManager.loadModel("Models/escavator/Excavator.obj");
    escavadora.scale(0.005f, 0.005f, 0.005f);
    escavadora.move(-30, 5, -50);

escavadora.setShadowMode(RenderQueue.ShadowMode.CastAndReceive);
    rootNode.attachChild(escavadora);
  }
```

A la hora de crear la grúa hemos echo algo muy similar a crear los otros elementos de la escena, la única diferencia es que en este caso, todos los elementos de la grúa de penden de otros elementos de la grúa, para que así sea mucho mas sencillo realizar los movimientos. Cada parte de la grúa se denomina nodo y cada nodo se le asigna a otro nodo.

A la hora de asignar estos nodos es sencillo, vamos asignándolos desde el que no tiene ningún nodo que dependa de el hasta el que mas tenga. Por los vamos asignando en este orden: gancho, carro, pluma y torre. A estos nodos también le damos una posición concreta y un a escala al igual que todos los elementos de la escena.

Constructor de la clase Grúa

```
public Grua(AssetManager assetManager){
    carro = new Node();
    carro = (Node)assetManager.loadModel("Models/carro.obj");
    carro.scale(0.5f);
    carro.move(0, 38f, 0);
    carro.setShadowMode(RenderQueue.ShadowMode.CastAndReceive);

    gancho = new Node();
    gancho = (Node)assetManager.loadModel("Models/gancho.obj");
    gancho.scale(0.5f);
    gancho.move(-10, 38f, 0);
    gancho.setShadowMode(RenderQueue.ShadowMode.CastAndReceive);

    pluma = new Node();
    pluma = (Node)assetManager.loadModel("Models/pluma_centro.obj");
    pluma.move(0.8f, 0f, -0.6f);
    pluma.scale(0.5f);
    pluma.setShadowMode(RenderQueue.ShadowMode.CastAndReceive);

    torre = new Node();
    torre = (Node)assetManager.loadModel("Models/torre.obj");
    torre.scale(0.5f);
    torre.setShadowMode(RenderQueue.ShadowMode.CastAndReceive);

    //gancho.attachChild(caja);
    carro.attachChild(gancho);
    pluma.attachChild(carro);
    this.attachChild(pluma);
    this.attachChild(torre);
}
```

4.3.2.- Control de los gráficos 3D

A la hora de poder hacer movimientos en jMonkey Engine 3 es algo tedioso, ya que teneos que insertar un controlador para cada movimiento. Este controlador enreda de la clase abstracta

AbstractControl, de la cual tenemos que sobre escribir algunos metodos, pero el que mas nos interesa es *protected void controlUpdate(float tpf)* ya que es en el que le diremos que movimiento queremos que haga nuestro nodo y en que direccion mediante un vector.

En el caso de la rotacion utilizamos la funcion rotate, a la cual le pasamos un vector con la velocidad a la cual queremos que gire.

Controlador de rotación

```
public class ControlRotar extends AbstractControl{

  private float speed;


  public ControlRotar(float speed){
    this.speed = speed;
  }
  public ControlRotar(){
  }

  @Override
  protected void controlUpdate(float tpf) {

    spatial.rotate(0, tpf=speed, 0);

  }

  @Override
  protected void controlRender(RenderManager rm, ViewPort vp) {

  }

  @Override
  public Control cloneForSpatial(Spatial spatial) {
    ControlRotar control = new ControlRotar();
    control.setSpeed(speed);
    control.setSpatial(spatial);
    return control;
  }

  public float getSpeed() {
    return speed;
  }

  public void setSpeed(float speed) {
```

```
    this.speed = speed;
  }
}
```

Para el caso de subir el gancho utilizamos la función move a la cual le pasamos un vector con la velocidad a la cual queremos que se mueva.

Control de subir el gancho

```
public class ControlGanchoSubir extends AbstractControl {

  private float speed;


  public ControlGanchoSubir(float speed){
    this.speed = speed;
  }
  public ControlGanchoSubir(){
  }

  @Override
  protected void controlUpdate(float tpf) {

    spatial.move(0, tpf, 0);

  }

  @Override
  protected void controlRender(RenderManager rm, ViewPort vp) {

  }

  @Override
  public Control cloneForSpatial(Spatial spatial) {
    ControlGanchoSubir control = new ControlGanchoSubir();
    control.setSpeed(speed);
    control.setSpatial(spatial);
    return control;
  }

  public float getSpeed() {
    return speed;
  }

  public void setSpeed(float speed) {
```

```
    this.speed = speed;
  }

}
```

Para el caso de mover el carro también utilizamos la función move a la cual le pasamos un vector con la velocidad a la cual queremos que se mueva. Como podemos ver la única diferencia entre los controladores de subir el gancho y mover el carro es unicamente el vector que le asignamos para el movimiento. En el caso de que ese controlador se quisiese utilizar para otro nodo, se podría utilizar perfectamente sin inconveniente ninguno.

Control de mover el carro hacia delante

```
public class ControlCarroDelante extends AbstractControl {
  private float speed;


  public ControlCarroDelante(float speed){
    this.speed = speed;
  }
  public ControlCarroDelante(){
  }

  @Override
  protected void controlUpdate(float tpf) {

    spatial.move(-tpf, 0, 0);

  }

  @Override
  protected void controlRender(RenderManager rm, ViewPort vp) {

  }

  @Override
  public Control cloneForSpatial(Spatial spatial) {
    ControlCarroDelante control = new ControlCarroDelante();
    control.setSpeed(speed);
    control.setSpatial(spatial);
    return control;
  }
```

```
  public float getSpeed() {
    return speed;
  }

  public void setSpeed(float speed) {
    this.speed = speed;
  }
}
```

4.3.3.- Mover los elementos gráficos 3D

Una vez tenemos los controladores y queremos hacer que cualquier parte de nuestra grúa se mueva tan solo tenemos que añadirle a nuestro nodo el controlador correspondiente para que haga el movimiento adecuado. En el caso de querer parar ese movimiento lo que hacemos es eliminar ese controlador de nuestro nodo.

Parar los movimientos de la grúa

```
public void pararMovimiento(){

      grua.getPluma().removeControl(ControlRotar.class);
      grua.getGancho().removeControl(ControlGanchoBajar.class);
      grua.getGancho().removeControl(ControlGanchoSubir.class);
      grua.getCarro().removeControl(ControlCarroAtras.class);
      grua.getCarro().removeControl(ControlCarroDelante.class);

      moviendo = false;
      System.out.println("ORDEN - Parando");
}
```

Siempre antes de ejecutar cualquier instrucción que no sea *pararMovimiento()* se ejecuta esta función, para asegurarnos de que la grúa esta parada antes de hacer otro movimiento, ya que por ejemplo si la grúa se encuentra rotando hacia la derecha y le decimos que rote a la izquierda la grúa se parara pro como esta hecho los controladores, o en el caso de que se asignara otra vez el mismo movimiento la grúa giraría mas rápido.

En el caso de la rotación es el único que nos permite crear solo un controlador para el movimiento. Para que gire en un sentido u otro lo que hacemos es asignarle una velocidad negativa o positiva dependiendo de si es derecha o izquierda.

Rotación de la pluma

```
public void rotarDerecha(){
      if(moviendo){
          pararMovimiento();
      }
      moviendo=true;
      System.out.println("ORDEN - Moviendo derecha");
      grua.getPluma().addControl(new ControlRotar(-0.0005f));
}

public void rotarIzquierda(){
      if(moviendo){
          pararMovimiento();
      }
      moviendo=true;
      System.out.println("ORDEN - Moviendo izquierda");
      grua.getPluma().addControl(new ControlRotar(0.0005f));

}
```

En el caso de los movimientos del gancho y del carro, se tienen en cuenta unos limites físicos, los cuales se van comprobando cada vez que movemos nuestro nodo. De esta forma el gancho y el carro no tomaran posiciones incorrectas respecto a la grúa.

Movimientos del gancho

```
public void subir(){
      if(moviendo){
          pararMovimiento();
      }
      moviendo=true;
      System.out.println("ORDEN - Subiendo");

      if(limiteGanchoSubir()){
        grua.getGancho().addControl(new ControlGanchoSubir(0.5f));
      }
}
```

Para calcular los limites de altura del gancho y el recorrido del carro se comprueba que la posición sea inferior a la de limite.

Comprobar los limites de altura del gancho

```
/**
* Comprueba si el gancho puede subir.
* @return
*/
public boolean limiteGanchoSubir(){
    boolean resultado = false;
    if(grua.getGancho().getLocalTranslation().getY() < 58.f){
      resultado = true;
    }
                        System.out.println("Altura    gancho:    "    +
grua.getGancho().getLocalTranslation().getY());
    return resultado;
}

/**
* Comprueba si el gancho puede bajar.
* @return
*/
public boolean limiteGanchoBajar(){
    boolean resultado = false;
    if(grua.getGancho().getLocalTranslation().getY() > -90.f){
      resultado = true;
    }
                        System.out.println("Altura    gancho:    "    +
grua.getGancho().getLocalTranslation().getY());
    return resultado;
}
```

Comprobar los limites de desplazamiento del carro

```
/**
* Comprueba si el carro puede moverse hacia delante.
* @return
*/
public boolean limiteCarroAdelante(){
     boolean resultado = false;
     if( grua.getCarro().getLocalTranslation().getX() > -90.f){
       resultado = true;
     }
                                System.out.println("Posicion    carro:
"+grua.getCarro().getLocalTranslation().getX());
     return resultado;
```

```
}

/**
* Comprueba si el carro puede moverse hacia atrás.
* @return
*/
public boolean limiteCarroAtras(){
      boolean resultado = false;
      if( grua.getCarro().getLocalTranslation().getX() < 5.f){
        resultado = true;
      }
                                    System.out.println("Posicion   carro:
"+grua.getCarro().getLocalTranslation().getX());
      return resultado;
}
```

Movimientos del carro

```
/**
* Mueve la grúa hacia delante
*/
public void adelante(){
      if(moviendo){
          pararMovimiento();
      }
      moviendo=true;
      System.out.println("ORDEN - Moviendo adelante");
      if(limiteCarroAdelante()){
        grua.getCarro().addControl(new ControlCarroDelante(0.05f));
      }
}


/**
* Mueve la grúa hacia detrás
*/
public void atras(){
      if(moviendo){
          pararMovimiento();
      }
      moviendo=true;
      System.out.println("ORDEN - Moviendo atras");
      if(limiteCarroAtras()){
```

```
        grua.getCarro().addControl(new ControlCarroAtras(0.05f));
      }
}
```

4.4.- Persistencia

Los usuarios del servidor se guardan en un fichero llamado usuarios.xml, para esto se utiliza un método conocido como JAXBuilder, tanto para la lectura como la escritura. Este método se basa en una serie de etiquetas para configurar los elementos del fichero y como se van a representar.

Lectura del xml con los usuarios

```
public List<DatosConexion> leerUsuarios(){
    GruaP grua=null;
    try {
      JAXBContext context = JAXBContext.newInstance(GruaP.class);
      Unmarshaller unmarshaller = context.createUnmarshaller();
                          grua   =   (GruaP)unmarshaller.unmarshal(new
File("usuarios.xml"));

    } catch (JAXBException ex) {
        Logger.getLogger(Ventana.class.getName()).log(Level.SEVERE,
null, ex);
    }
    return grua.getUsuarios();
}
```

Escritura del xml de los usuarios

```
public void escribirUsuarios(){
    try {
      JAXBContext contex = JAXBContext.newInstance(GruaP.class);
      Marshaller marshaller = contex.createMarshaller();
      GruaP grua = new GruaP();
      grua.setUsuarios(this.usuarios);
           marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
true);
      marshaller.marshal(grua, new FileWriter("usuarios.xml"));
      marshaller.marshal(grua, System.out);

    } catch (JAXBException ex) {
```

```
        Logger.getLogger(Ventana.class.getName()).log(Level.SEVERE,
null, ex);
    } catch (IOException ex) {
        Logger.getLogger(Ventana.class.getName()).log(Level.SEVERE,
null, ex);
    }
}
```

Para guardar las conexiones en la aplicación android, se utiliza un sistema de base de datos llamado SQLite. En la clase Persistencia tenemos todos los métodos necesarios para leer, crear y eliminar los usuarios.

Persistencia para SQLite

```
public class Persistencia {
  SQLiteDatabase db;

  public Persistencia(Context context){
       ConexionSQLiteHelper dbh = new ConexionSQLiteHelper(context,
"datosConexion", null, 6);
    db=dbh.getWritableDatabase();
  }

  public List<Conexion> getDatosConexion(){
    List<Conexion> lista = new ArrayList<Conexion>();
    Cursor c= db.rawQuery("SELECT * FROM conexiones;", null);
    if(c.moveToFirst()){
      do{
        lista.add(cursorToConexion(c));
      }while(c.moveToNext());
    }
    return lista;
  }

  public Conexion cursorToConexion(Cursor cursor){
    Conexion conexion = new Conexion();
    conexion.setDireccion(cursor.getString(0));
    conexion.setUsuario(cursor.getString(1));
    conexion.setClave(cursor.getString(2));
    conexion.setNombre(cursor.getString(3));
                  Log.i("PERSISTENCIA", "Leyendo conexion: " +
conexion.toString());
    return conexion;
```

```
  }
  /*public Integer getId(){
    List<Conexion> lista= getDatosConexion();
    Conexion = lista.get(lista.size());
  }*/

  public void addConexion(Conexion conexion){
    db.execSQL("INSERT INTO conexiones " +
        "VALUES('"+conexion.getDireccion()+"', " +
           "'"+conexion.getUsuario()+"', " +
           "'"+conexion.getClave()+"', " +
           "'"+conexion.getNombre()+"');");
                    Log.i("PERSISTENCIA",   "Guardando    conexion:
"+conexion.toString());
  }

  public void removeConexion(Conexion conexion){
    db.execSQL("DELETE FROM conexiones WHERE " +
        "direccion='"+conexion.getDireccion()+"'" +
        "and usuario='"+conexion.getUsuario()+"'" +
        "and clave='"+conexion.getClave()+"'" +
        "and nombre='"+conexion.getNombre()+"'");
  }
}
```

5.- Glosario de términos

- API (Application Programming Interfaces): Conjunto de rutinas, protocolos y herramientas para construir aplicaciones.
- Asset: Es cualquier elemento utilizado para la creación de animaciones.
- Iluminación: La iluminación permite que los objetos se vean brillantes en la dirección de la luz y oscuros en el lado opuesto.
- Antialiasing: consiste en la eliminación de la información de frecuencia demasiado elevada para poder ser representada
- Color Depth: Se refiere a la cantidad de bits de información necesarios para representar el color de un pixel en una imagen digital.
- gamma correction: es como se denomina cierta operación no lineal que se usa para codificar y decodificar luminancia o valores triestimulos en sistemas de video o imagen.
- VSync: La sincronización vertical detecta la tasa de refresco de tu monitor (60Hz, por ejemplo) y limita la cantidad de fotogramas por segundo (FPS) con los que el juego se mueve a una cantidad equivalente a esta tasa de refresco.

6.- Bibliografía

- Juegos 3D en java: Blender y jMonkey Engine
  https://www.adictosaltrabajo.com/tutoriales/blenderava-monkey-engine/
- Foro de jMonkey Engine: https://hub.jmonkeyengine.org/
- Documentación java para JMonkeyEngine3: http://javadoc.jmonkeyengine.org/
- Grúas torre: http://www.monografias.com/trabajos32/grua-torre/grua-torre.shtml
- https://jorgeluh.wordpress.com/2008/10/09/agua-animada-con-blender-y-java-3d-parte-1/
- https://jorgeluh.wordpress.com/2008/10/19/agua-animada-con-blender-y-java-3d-parte-2/
- Importar de Blender a jMonkey Engine:
  https://jmonkeyengine.github.io/wiki/sdk/blender.html
- https://github.com/jMonkeyEngine/examplegame-skulls
- Joystick: http://www.instructables.com/id/A-Simple-Android-UI-Joystick/
- Apuntes sobre diseño blender. Facilitados por Dionisio.
- Apuntes sobre las curvas de Bezier. Facilitados por Dionisio.

7.- Apéndices

- Anteproyecto - Miguel Angel Gonzalez Hernandez - Grúa Torre.pdf