Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño y programación con Unity3D de un avatar interactivo con sincronización musical

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: José Juan Preciado Sánchez

Tutor: Adolfo Muñoz García

2014/2015

Resumen

Este proyecto trata de proporcionar un mecanismo para la ejecución de actuaciones musicales compuesto por un director y diversos interpretes. Es un instrumento para la enseñanza musical de alumnos de la ESO.

Se ha desarrollado una aplicación para el director que hace de servidor para los intérpretes. Se puede seleccionar avatares que representen a cada intérprete así como configurar las respuestas a las distintas acciones de cada uno, pudiendo asignar generadores de nuevos sonidos, modificaciones de aspectos visuales o variaciones de efectos auditivos.

Para los intérpretes se ha generado una aplicación *smartphone* la cual permite conectarse a la aplicación servidora y acceder a los diversos paneles de controles para la actuación.

Palabras clave: *Unity3D*, avatar, OSC, música, educación.

Índice de contenido

Contenido

Índice de ilustraciones

Índice de tablas

1. Introducción

1.1. Propósito

El proyecto consistirá en la realización de un mecanismo que facilite la ejecución de piezas musicales en grupo, a modo de orquesta, mediante el uso de *smartphones* y un ordenador. Para lo que se realizarán dos aplicaciones.

La primera de ellas, la proporcionada a cada "interprete" en su *smartphone*, será la que facilite los manejadores y controles necesarios -a los cuales se denominará como instrumentos a partir de ahora- para la manipulación de efectos y generación de sonidos, ésta les permitirá configurar la conexión de red con el servidor, seleccionar entre distintos paneles y, por último, modificar los estados de los distintos controles enviando mensajes que serán traducidos en sonidos y efectos visuales y sonoros.

La segunda será la encargada de la reproducción musical y efectos visuales de cada una de las órdenes recibidas por los distintos interpretes. Se pretende que ésta aplicación permita al director realizar una configuración individual de los clientes conectados, asignando los efectos que se producirán o sonidos que se generarán al detectar la activación de los diversos controles por parte de los intérpretes. Estos efectos podrán apreciarse tanto sonora, se mezclarán todos los resultados en una salida de audio común, como visualmente, mediante los avatares que el usuario del servidor asignará a cada uno de los diferentes intérpretes. Controlará que los efectos de audio sean de asignación única debido a la salida común resultante.

Este proyecto se enmarca en la educación musical como instrumento para alumnos de la ESO.

1.2. Motivación personal

La motivación inicial fue la posibilidad de trabajar con la herramienta de desarrollo de videojuegos multiplataforma Unity3D, hacer uso del lenguaje de programación *C# y* consolidar conocimientos del uso de redes.

Una vez realizada la primera reunión con el tutor y conocido con mayor detalle la idea general fueron surgiendo nuevas motivaciones, como el uso del protocolo OSC para la comunicación, que tras la búsqueda de información se observó la simplicidad y eficacia que puede presentar, y el desarrollo de una aplicación para dispositivos *Android*.

1.3. Antecedentes

Ilustración 1. SoundCool

La idea de este proyecto surge de la iniciativa *SoundCool*, que pretende ser una plataforma para la educación musical mediante móviles, *tablets*, *kinect* y *MAX/MSP/JITTER* (SoundCool).Está en estado de pruebas de campo en clases de música como un nuevo método de enseñanza para los alumnos.

Para el desarrollo de las distintas aplicaciones se pretende hacer uso de la herramienta de desarrollo multiplataforma *Unity3D, p*ara la comunicación entre "interpretes" y "director" se ha partido del código publicado por (Heavers), el cual se basa en el protocolo de comunicación *Open Sound Control* (OSC) y para el sintetizador de audio.

1.4. Objetivos

Los objetivos para este proyecto se dividirán en dos partes, por un lado para la aplicación móvil, que será la que ejerza el papel de instrumento musical y por otro lado la de ordenador, encargada de organizar a cada uno de los intérpretes, reproducir los sonidos y efectos visuales.

Objetivos para la aplicación móvil:

- Desarrollo de interfaz intuitiva
- Presentación de diferentes instrumentos
- Conexión con la aplicación que ejerce de servidor

Objetivos a priori para la aplicación de ordenador:

- Conexión a red como servidor
- Soporte dinámico del número de clientes
- Asignación de un avatar por cliente
- Generadores de sonidos
- Efectos de audio de asignación única
- Efectos de video
- Posibilidad de asignar los generadores de audio y efectos a los controles de los instrumentos
- Configuración de instrumentos única para cada cliente, un mismo instrumento podrá tener diferentes efectos en distintos clientes
- Interfaz que facilite el control de las distintas funcionalidades
- Reproducción de generadores de sonidos, efectos de video y audio.

2. Herramientas empleadas

2.1. Unity3D

Ilustración 2. Logo Unity3D

Es un motor gráfico multiplataforma creado por *Unity Technologies* que está disponible como plataforma de desarrollo para *Microsoft Windows* y *OS X*. Es fácil de usar tanto para desarrollos en *2D* y *3D* y se actualiza constantemente con nuevos contenidos y opciones. En la versión 5.0 se han liberado un gran número de características hasta entonces solo disponibles en la versión profesional, entre ellas está la liberación del uso de las librerías *Network* para *Android*, que es necesario para la aplicación de los intérpretes. A continuación se presentarán algunas de las principales características de *Unity3D* como son:

- Exportación multiplataforma gratuita (móvil, *web, PC y Mac, PS3*, ...).
- Scripting en *Java Script*, *C#* ó *Boo*. Contiene un editor de código.
- Soporta gran cantidad de paquetes *3D* y texturas de múltiples extensiones.
- Soporte para creación de redes y aplicaciones en línea.
- Editor de Terrenos y vegetación incorporada.
- Creación de aplicaciones en *2D* y *3D*.
- Gestor de animaciones, arboles de mezcla y máquinas de estado.
- Disponible completamente gratis desde su sitio oficial.

2.2. Lenguaje de programación

Unity ofrece tres lenguajes a la hora de programar, estos son *UnityScript* (un *javascript* con ligeras modificaciones), *Boo* y *C#*, pudiéndose utilizar todos a la vez.

El lenguaje elegido ha sido *C#* por varios motivos. El principal motivo es el conocimiento previo de este, además del hecho de ser más fácil de depurar que un lenguaje de tipado dinámico como es *UnityScript*. Otro de los motivos es que la mayoría de la documentación está desarrollada en *C#* o *UnityScript* por lo que *Boo* queda descartado.

Ilustración 3. Microsoft .NET

C# es un lenguaje de programación orientado a objetos, desarrollado y estandarizado por *Microsoft* como parte de su plataforma *.NET* que más tarde fue aprobado como un estándar por la *ECMA (ECMA-334) e ISO (ISO/IEC 23270)*. Su sintaxis básica deriva de *C/C++* y utiliza el modelo de objetos de *.NET*, similar al de *Java*, aunque con mejoras derivadas de otros lenguajes (Herraez, 2013/2014).

2.3. Introducción a *Unity*

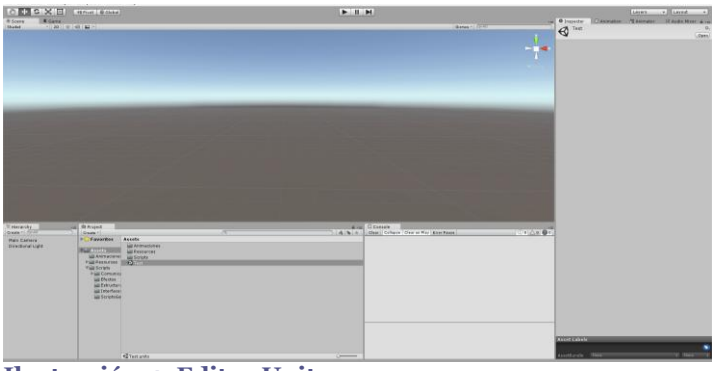


Ilustración 4. Editor Unity

El elemento básico en *Unity* es el *GameObjecto* que es la clase principal en la cual se añaden el resto de atributos como componentes, *scripts,* fuentes de audio, cámaras, etc . La forma que se utiliza para organizar es mediante escenas las cuales representan los distintos niveles de la aplicación, desde los menús hasta los créditos.

Los objetos de una escena son creados al cargarla y destruidos al cambiar a otra, por lo que para conservar alguno de los objetos entre escenas hay que especificarlo por código.

Algunos de los componentes más importantes de los *GameObjects* son:

- El *renderer*, que se encarga de que el objeto sea visible, dándole una forma y color o textura
- Los *rigidbody* y *collider*, encargados de gestionar las colisiones con otros elementos y, los *rigidbodys* en particular, las características físicas
- La cámara representa la visión que se tendrá de cada nivel
- El componente al que se agregan *scripts*

Algunos de los atributos de los componentes se pueden editar tanto por código como desde el editor como son las variables públicas de los *scripts* agregados. Todos los compontes pueden habilitarse o deshabilitarse excepto el *transform* que viene por defecto en todos los *gameobjects* y decide la posición, rotación y escala de cada objeto.

Para facilitar la creación y multiplicación de objetos se pueden realizar *prefabs* a los cuales se les asigna un objeto configurado previamente para luego poder instanciarlos por código o desde la ventana de edición de *Unity3D*.

El ciclo de vida que siguen los *gameobjects* se basa en el ciclo de vida clásico de los motores gráficos se inicializan por primera vez y luego entran en un bucle hasta que es destruido, dentro del bucle se realizan las actualizaciones físicas, de código o animaciones y renderizado del objeto hasta que se destruye.

Para controlar las diferentes etapas de vida se hace uso de la librería ofrecida por *Unity3D* llamada *UnityEngine* y los *scripts* tienen que heredar de la clase *MonoBehaviour* lo que permite acceder a los siguientes métodos como explica (Unity Technologies):

Editor

- *Reset*: Para inicializar las propiedades de los *scripts* cuando es añadido por primera vez a un objeto y cuando el comando *Reset* es usado.

Primera carga de escena

- *Awake*: Siempre es llamada antes de cualquier función *Start* y antes de que cualquier *prefab* sea instanciado.
- *OnEnable*: Solo se llama si el objeto está activo, es llamada justo cuando un objeto es habilitado. Esto ocurre cuando es creada una instancia de *MonoBehaviour*, cuando un nivel es cargado o un objeto con el *script* como componente es instanciado.

Antes de la actualización del primer *frame*

- *Start*: es llamada justo antes de la actualización del primer *frame* solo si la instancia del *script* está habilitada.

Entre *frames*

- *OnApplicationPause*: es llamada al final del *frame* donde se ha detectado la pausa, un *frame* adicional será realizado mostrando el estado de pausa de la aplicación.

Orden de actualización

- *FixedUpdate*: Es llamado más frecuentemente que *Update* puede ser llamado varias veces por *frame* si el refresco es lento y no será llamado entre *frames* si el ratio de actualización es alto. Es donde se asignan las instrucciones de actualización de físicas por su independencia al ratio de actualización de *frames*.

- *Update*: Es llamado una ver por cada *frame*.

- *LateUpdate*: es llamado después de cada *Update*. Es usado habitualmente para gestionar los movimientos de la camara.

Renderizado

- *OnPreCull*: El primer evento de renderizado, en el que se determina que objetos son visibles para la camara.

- *OnBecameVisible/OnBecameInvisible*: usada cuando un objeto se vuelve visible o invisible para alguna cámara.

- *OnWillRenderObject*: lanzada una vez por cada cámara si el objeto es visible.

- *OnPreRender*: ejecutada antes de que la cámara renderice la escena.

- *OnRenderObject*: tras todos los renderizados regulares se puede usar este método para dibujar geometrias personalizadas mediante las clases *GL* o *Graphics*.

- *OnPostRender*: una vez se ha finalizado el renderizado.

- *OnRenderImage*: usado para el procesado de la imagen resultante.

- *OnGUI*: llamada múltiples veces por cada *frame* en respuesta a los eventos *GUI* (interfaz gráfica del usuario).

- *OnDrawGizmos*: usada para dibujar *Gizmos* en la escena.

Co-rutinas

- *yield*: las corutinas siguen tras la llamada de todas las funciones de actualización del siguiente *frame*.

- *yield WaitForSeconds*: continua tras un tiempo especifico.

- *yield WaitForFixedUpdate*: continua tras todos los *FixedUpdate* hayan sido llamados en todos los *scripts*.

- *yield WWW*: continua tras la descarga de una web.

- *yield StartCoroutine*: encadena la corutina y esperará hasta el fin de la corutina anterior.

Cuando el objeto es destruido

- *OnDestroy*: esta función es llamada tras todas las actualizaciones del último *frame* en el que el objeto es destruido.

Cuando se sale de la aplicación

- *OnApplicationQuit*: es llamada en todos los objetos antes de que la aplicación sea quitada, en las aplicaciones web es llamada cuando la vista de la página es cerrada.

- *OnDisable*: es activada cuando el objeto pasa a deshabilitado o inactivo.

En la siguiente imagen se muestra un esquema resumen de todo lo explicado con anterioridad y las posibles transiciones que se pueden presentar.

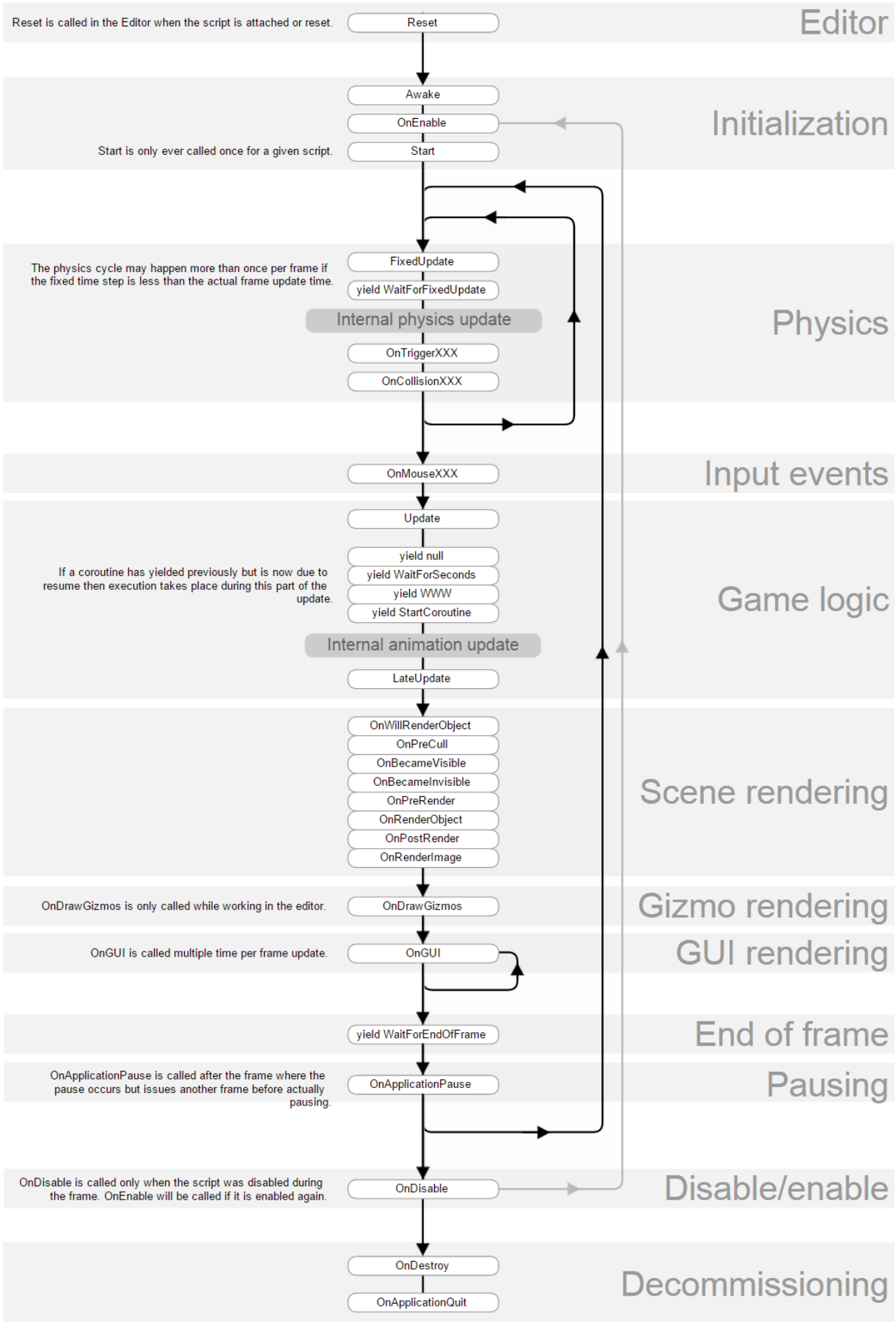


Ilustración 5. Orden de ejecución

2.4. OSC

Ilustración 6. OSC

Open Sound Control es un protocolo de comunicaciones que permite comunicar instrumentos de música, computadoras y otros dispositivos multimedia (por ejemplo móviles o *PDA's* equipados con *bluetooth*) pensado para compartir información musical en tiempo real sobre una red.

Aparece como reemplazo del *MIDI*, siendo muy superior en características y capacidades.

Características principales del protocolo:

- Ampliable, dinámico. Esquema de nombres simbólicos tipo URL
- Datos numéricos simbólicos y de alta resolución
- Lenguaje de coincidencia de patrones (*pattern matching*) para especificar múltiples receptores de un único mensaje
- Marcas de tiempo (time *tags*) de alta resolución.
- Mensajes "empaquetados" para aquellos eventos que deben ocurrir simultáneamente
- Sistema de interrogación para encontrar dinámicamente las capacidades de un servidor OSC y obtener documentación.

Puede ser transportado por varios protocolos, comúnmente se usa UDP. (Fundación Wikimedia, Inc.)

2.5. FMOD

Ilustración 7. Logo FMOD

Es una herramienta de desarrollo de sonidos para videojuegos y aplicaciones desarrollado por la empresa *Firelight Technologies*, que reproduce y mezcla ficheros de audio de diversos formatos en diferentes sistemas operativos. Es de libre descarga y uso para todo tipo de proyecto (Rose). Son las librerías que usa *Unity* para todo lo relacionado con el audio espacial y ofrece a los usuarios una serie de herramientas como los *Audio Source, Audio listener* y *Mixer Audio* para la reproducción además de muchos otros efectos entre los que se encuentran el eco, reverberación, distorsión...

3. Análisis

3.1. Planificación

Para lograr los distintos objetivos se seguirá la siguiente planificación divida en dos partes, la primera para la aplicación móvil y la segunda para la de ordenador y comunicaciones.

Planificación a priori para la aplicación móvil:

- Diseño de la interfaz y transiciones entre distintas ventanas
- Diagrama de *scripts*
- Pre-diseño de la estructura inicial de los mensajes
- Programación y pruebas de:
  - Transiciones
  - Pulsación de los distintos controles
  - Establecimiento de conexión con servidor de prueba
- Revisión del funcionamiento inicial

Planificación a priori para la aplicación de ordenador:

- Diagrama de *scripts*
- Generación de estructura para los clientes
- Gestor del conjunto de clientes conectados
- Pruebas de nuevos "interpretes"
- Rediseño de estructuras
- Diseño de la interfaz
- Primera implementación de la interfaz
- Reestructuración del código
- Estructura para relacionar la interfaz y los diferentes clientes
- Pruebas de ejecución
- Segunda implementación de la interfaz
- Pruebas de ejecución

3.2. Navegabilidad de los clientes

En el siguiente esquema se mostrara la diferentes ventanas por las que se puede navegar en la aplicación móvil así como las posibles transiciones de esta

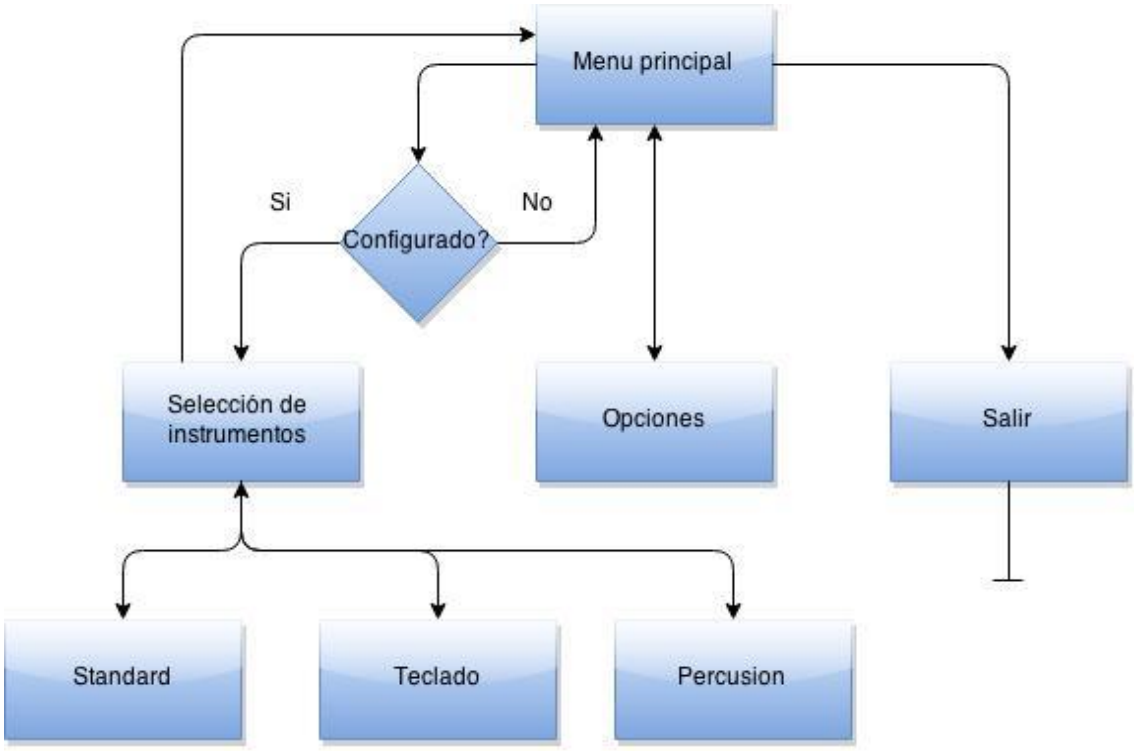


Ilustración 8. Navegación aplicación cliente.

Se puede observar una transición condicional desde el menú principal a la selección de instrumento, la cual hace referencia a que en un primer momento es necesario realizar la configuración inicial de las opciones y hasta que esta no haya sido realizada no se podrá acceder al menú de selección.

El menu principal desde el cual se puede salir de la aplicación, configurar las opciones de red o ir al menu de selección de instrumentos (Empezar)

Ilustración 9. Menú principal

recordar que no se podrá acceder al menu de selección de instrumentos sin una configuración previa de las opciones. Por lo que accediento a estas

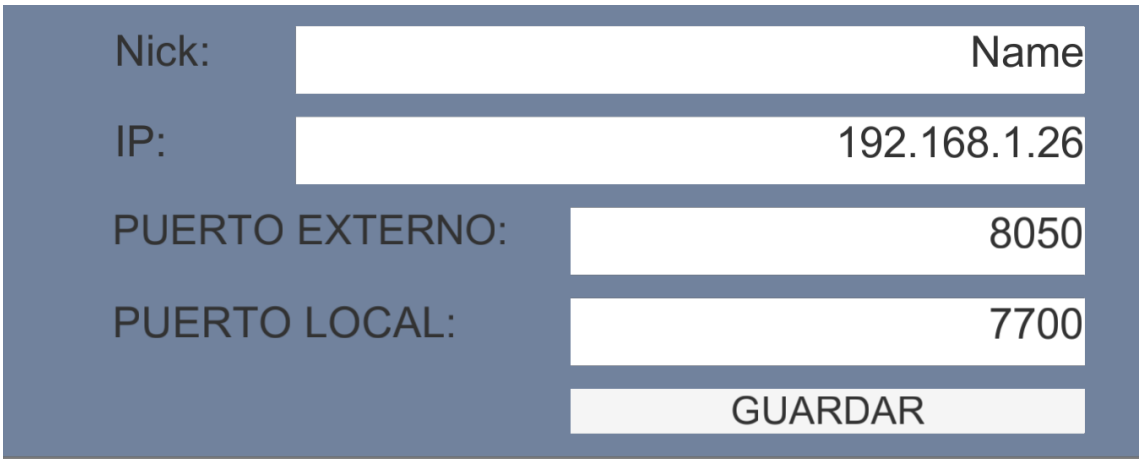


Ilustración 10. Opciones intérprete

se podrá configurar la dirección *IP*, puertos de entrada/salida y un *nick* identificativo para el director, una vez realizado nos devolverá al menú principal desde el cual se permitirá empezar y aparecerá una ventana de selección de instrumento

Ilustración 11. Selección instrumento

al seleccionar uno de ellos se mostrara los diferentes controles de dicho instrumento, los cuales se pueden observar a continuación

Standard

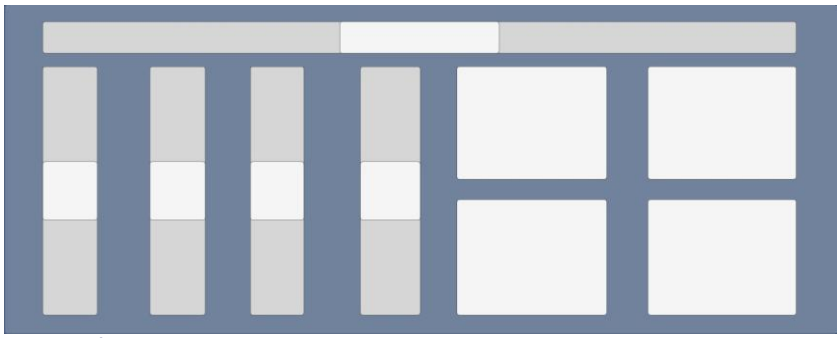


Ilustración 12. Standard

Teclado

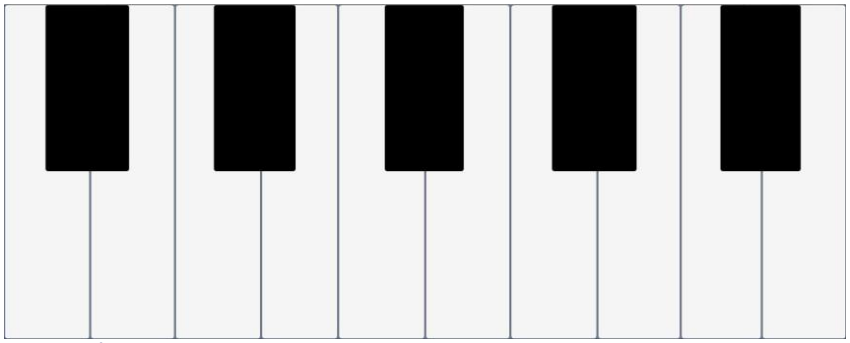


Ilustración 13. Teclado

Percusion

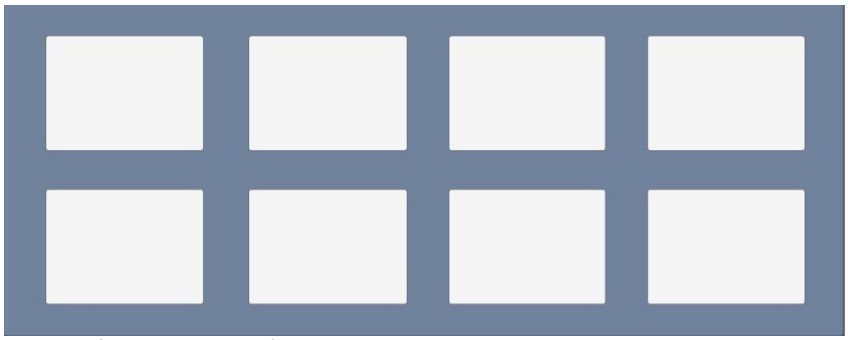


Ilustración 14. Percusion

3.3. Interfaz de los clientes

La interfaz que se ha realizado es sencilla y usa elementos predefinidos por *Unity* a los cuales se les ha modificado las dimensiones y textos para adaptarlos a nuestro caso de uso. Como se trata de elementos básicos sin mayores modificaciones realizadas se recogerán en la siguiente tabla los diferentes elementos empleados para cada una de las diferentes escenas

Escena	Texto plano	*Input Field*	Botón	*Scrollbar*
Menú principal	0	0	3	0
Opciones	4	4	1	0
Selección de instrumento	0	0	3	0
Standard	0	0	4	5
Percusion	0	0	15	0
Teclado	0	0	8	0
Total	4	4	34	5

Tabla 1. Interfaz cliente

Cada uno de estos elementos tiene sus propias características que se mostraran a continuación:

Características comunes

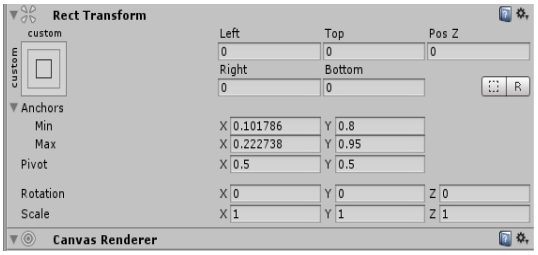


Ilustración 15. Componentes comunes interfaz

Todos los elementos pertenecientes a un *Canvas* -que es el encargado por *Unity3D* para gestionar las interfaces- tienen como componentes un *Rect Transform* con algunas propiedades adicionales al *Transform* tradicional para ajustar el redimensionamiento de estos elementos, además se añade la componente *Canvas Renderer* para dibujar estos componente dentro del *Canvas que los contiene.*

Texto plano

Los textos incorporan un componente denominado *Text* en el cual se puede configurar el texto que muestra, las fuentes que se van a usar, estilo, tamaño... propiedades de alineación, comportamiento a seguir en caso de extenderse fuera de los limites y el color o material asignados.

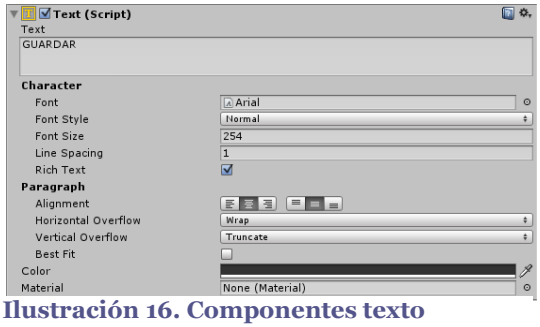


Ilustración 16. Componentes texto

Input Field

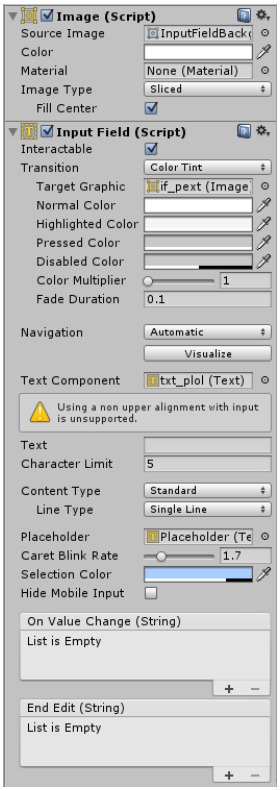


Ilustración 17. Componentes Input Field

Estos elementos incorporan dos textos, uno para mostrar cuando no se ha insertado ningún texto (*Placeholder*) y otro con el que se va introduciendo y luego se recogerá mediante scripts (*txt_ip*).

Además en las propiedades del objeto principal se encuentran los siguientes componentes *Image* e *Input field*.

Image es la que carga el fondo blanco tras los textos e Input *Field* es el componente principal que incorpora la funcionalidad principal. En este componente se puede seleccionar entre otros los colores que irá adoptando según la situación, los textos que forman parte de él y configurar distintas propiedades respecto a los textos como puede ser la limitación a una cierta cantidad de caracteres.

También incorpora la posibilidad de asignar ciertas funciones públicas dependiendo de si se está editando o se terminó de hacerlo, lo cual se ha preferido hacer directamente desde el código.

Botón

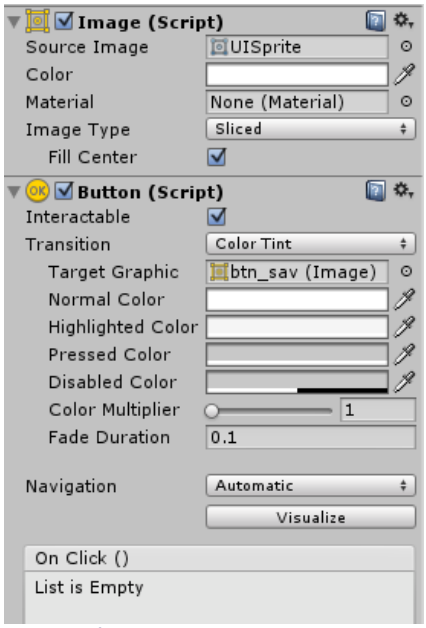


Ilustración 18. Componentes botón

Los botones incorporan un texto que muestran la funcionalidad que tienen asignada.

El objeto principal tiene dos componentes adicionales a los compartidos por los elementos de interfaz que son *Image* y *Button*,

El componente *Image* al igual que en *Input Field* se usa para establecer el color de fondo del botón y el componente *Button* incorpora la funcionalidad propia del botón en la que se puede marcar si se puede interaccionar o no con él, la imagen que está usando de fondo y los diferentes colores que irá cobrando dependiendo de la situación

Al igual que en *Input Field* ofrece la opción de incorporar funcionalidades al botón desde el editor en caso de que sea *clickeado* pero se ha preferido realizar el trabajo por *scripts* y declarar a la función privada.

Scrollbar

Los *scrollbar* están compuestos por dos elementos además del principal y son: *Handle* compuesto por una imagen es el que dibuja el botón, *Sliding Area* encargado de marcar el recorrido que puede realizar.

El objeto principal *Scrollbar* es el que incorpora la funcionalidad a todas ellas, al igual que los anteriores elementos mencionados incorpora un *Image* para dibujar el fondo del objeto y un componente llamado *Scrollbar* encargado de gestionar los elementos mencionados.

Proporciona opciones para activar o desactivar la interactividad, controlar los colores que cobra la imagen en función de la situación así como los posibles valores que puede tomar y la dirección y sentido que seguirá el *scroll*.

Como se ha podido observar *Unity3D* sigue la misma dinámica en todos sus elementos proporcionando un mecanismo para asignar funciones públicas de nuestros scripts directamente en las propiedades del objeto.

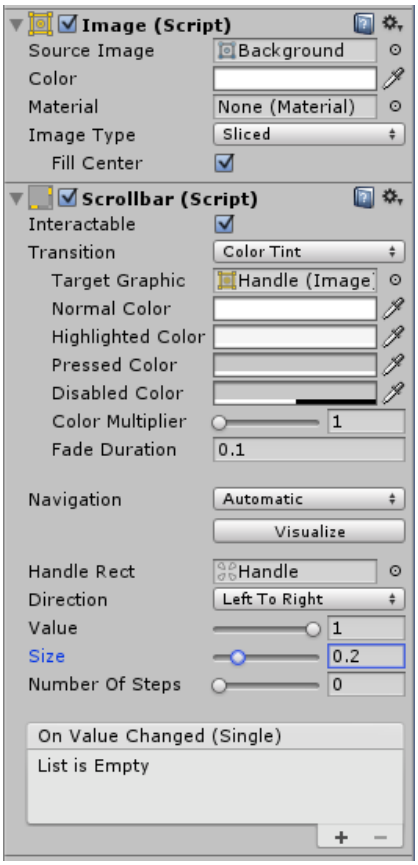


Ilustración 19. Componentes scrollbar

3.4. Funciones clientes

La funcionalidad presente en el cliente es muy básica, lo primero a realizar será configurar los diferentes apartados de la ventana de opciones, lo que al pulsar en guardar se configurarán los script de conexión red y se enviará un mensaje a la aplicación que hace de servidor con el *nick* introducido y la dirección *IP* del dispositivo.

Una vez establecida la conexión se habilitará la opción de Empezar y la que al ser activada se mostrará el menú de selección del instrumento, en el que se mostraran las distintas opciones que se pueden escoger y al seleccionar una se generará un nuevo mensaje informando al servidor de la opción seleccionada.

Realizado esto se cargará el panel de controles que al interactuar con él se irán enviando mensajes al servidor informando de las acciones del usuario.

3.5. Navegabilidad del servidor

La aplicación creada a modo de servidor presenta su complejidad en las funcionalidades más que en su navegabilidad la cual es sencilla.

Al iniciar la aplicación se presentará la siguiente ventana

Ilustración 20. Opciones servidor

en ella se pueden observar cuatro ventanas, entre ellas lo único que presenta cierta navegabilidad es el botón de empezar, el cual hace que desaparezcan las opciones y se muestren los avatares, y la ventana que está vacía.

La ventana vacía se activa al 'Fijar' el 'Número de interpretes' lo cual para un ejemplo de 3x3 presenta la siguiente estructura

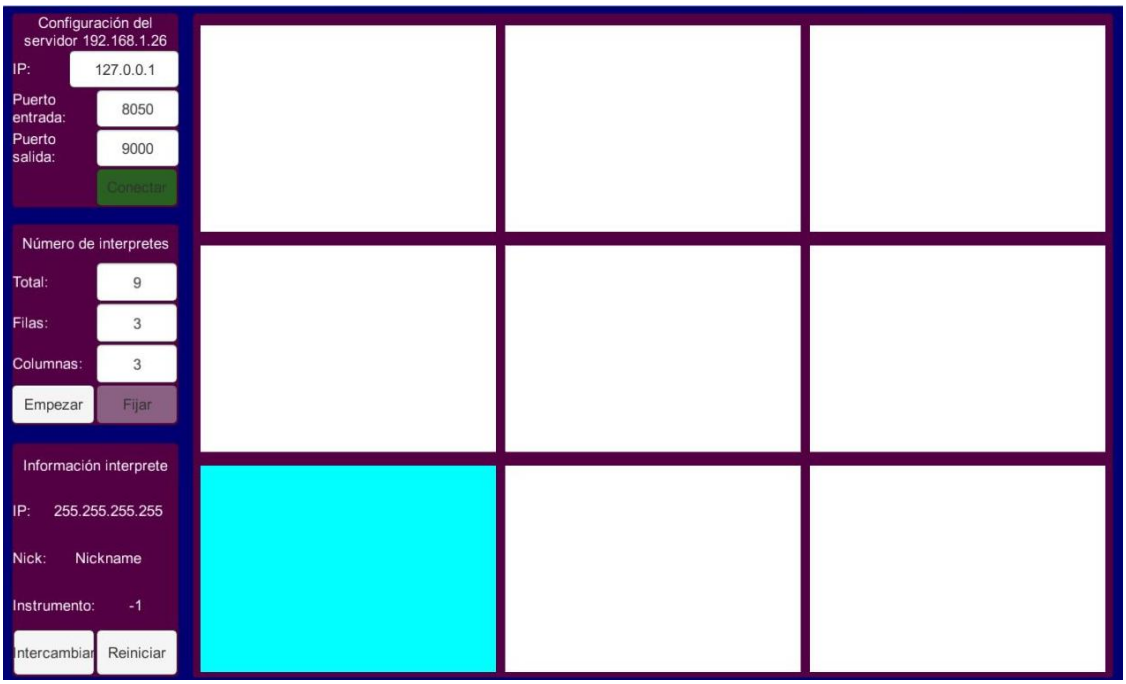


Ilustración 21. Ejemplo organización 3x3

cada uno de los botones representará a un intérprete, y los que estén de color *cyan* indican que en esa posición se ha conectado uno de los intérpretes.

Al hacer *click* sobre uno de estos botones en color *cyan* se mostrará en el panel de 'Información interprete' sus datos y, si además ya ha seleccionado un instrumento la ventana será reemplazada por la siguiente

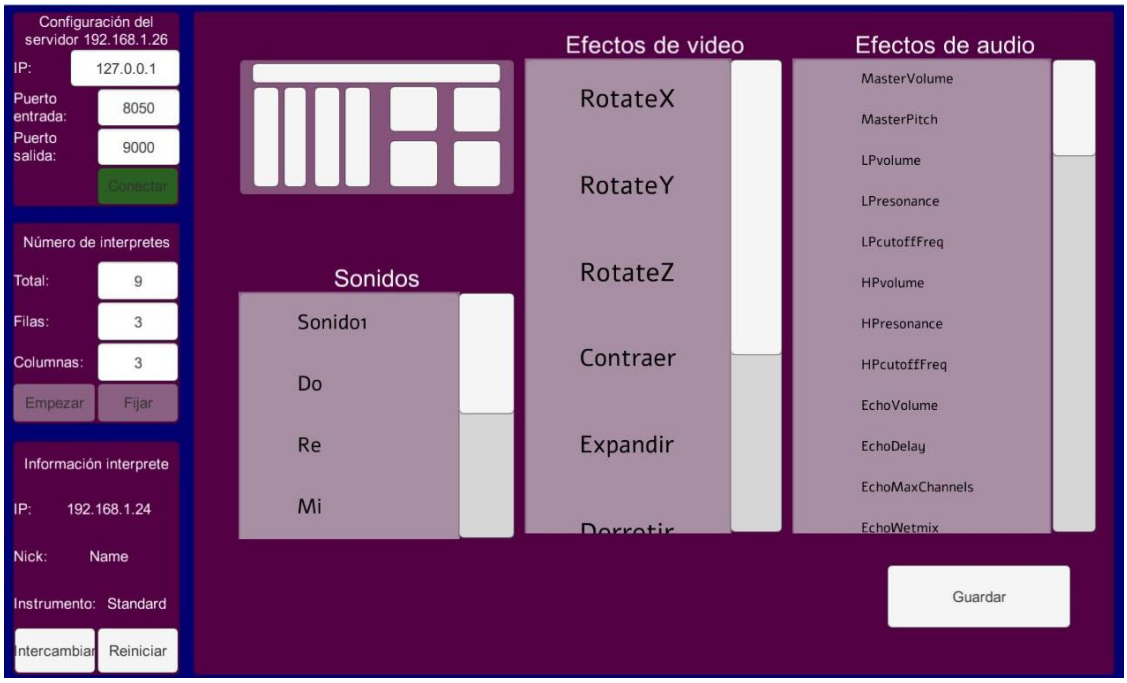


Ilustración 22. Configuración de interprete

en ella aparecerán las diferentes formas en las que puede ser configurado un intérprete y una representación del panel que ha seleccionado.

Al pulsar el botón de 'Guardar' volverá a la anterior ventana, mostrada en la Ilustración 21, donde se muestran los huecos disponibles y ocupados por los distintos interpretes.

Una vez configurados todos los intérpretes se podrá empezar pulsando dicho botón, lo cual ocultará el menú de opciones y mostrará los avatares de los distintos interpretes conectados.

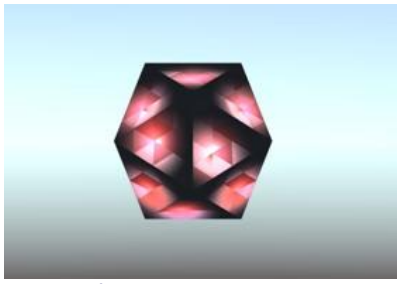


Ilustración 23. Avatar Interprete

además de lo ya presentado se han configurado las teclas *'Esc'* para cerrar la aplicación y la tecla *'M'* para volver a mostrar el menú de opciones.

3.6. Interfaz del servidor

La interfaz que se emplea para el servidor además de los elementos ya explicados para el cliente incorpora una serie de elementos de interfaces más avanzadas.

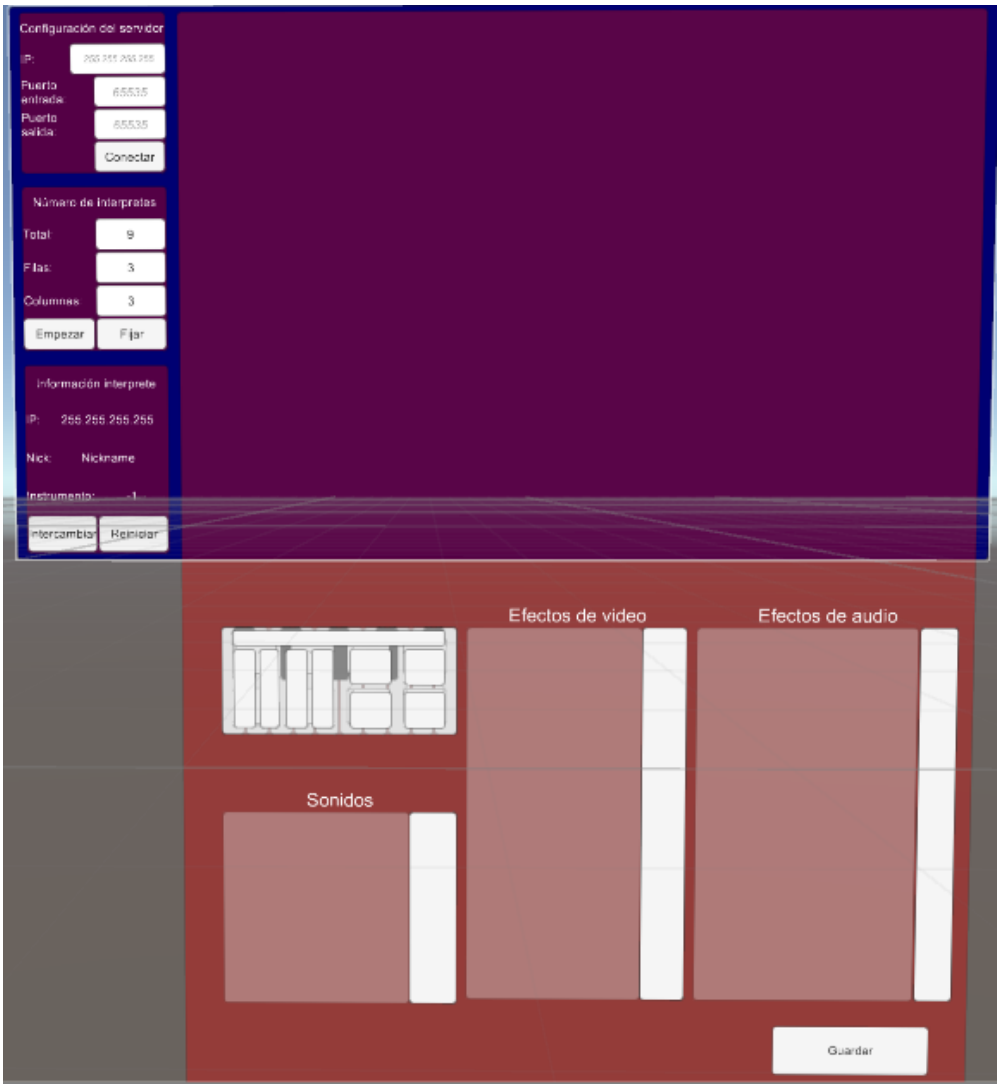


Ilustración 24. Interfaz servidor completa

En la anterior ilustración se muestra una imagen completa de la interfaz del servidor, al ser una única ventana con muchas funciones se irán comentando por separado.

Ilustración 25. Paneles Servidor

Para empezar tenemos los paneles de configuración del servidor, número de interpretes e información de intérpretes, estos no incorporan nada que no se haya explicado anteriormente ya que están compuestos por un panel y diversos elementos de texto plano, botones e *Input Field*.

Al otro lado de estos paneles se observa ese gran panel que sobresale y que una vez el programa está en ejecución puede tener dos posiciones distintas, una cuando se están gestionando los intérpretes y otra cuando se está configurando uno en particular.

Gestión de interpretes

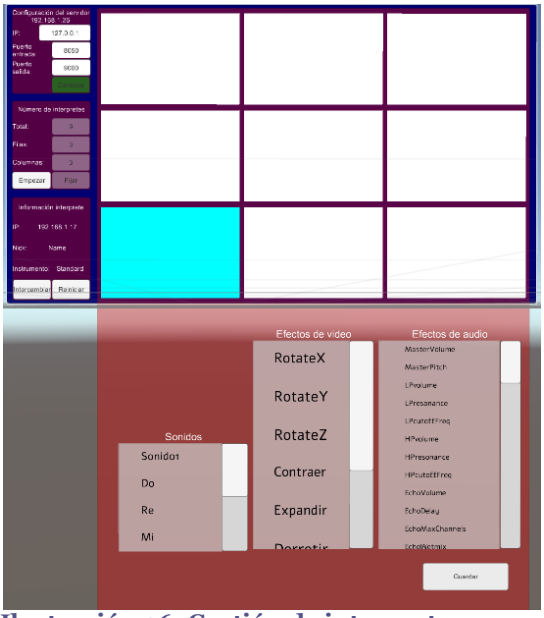


Ilustración 26. Gestión de interpretes

Configuración de interprete

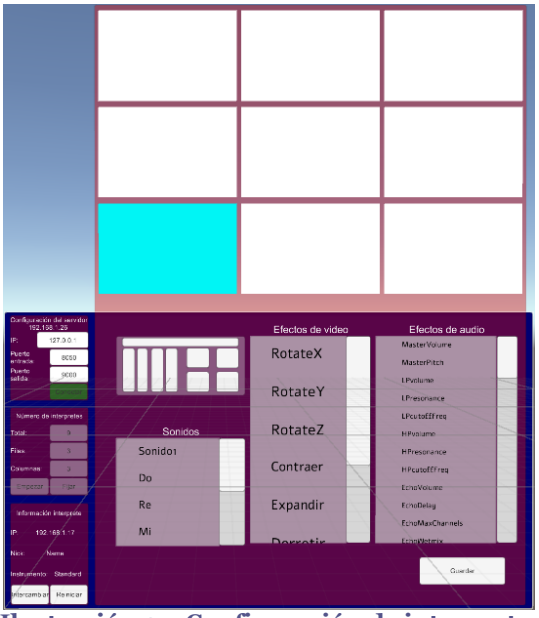


Ilustración 27. Configuración de interprete

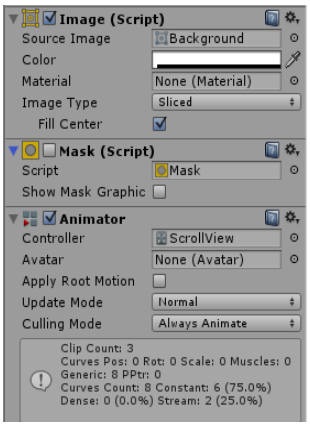


Ilustración 28. Componentes scrollview

Para conseguir este efecto lo primero ha sido añadir un objeto vacio al que se ha denominado *ScrollView* dentro del *Canvas* obteniendo así los componentes básicos, *Rect Transform* y *Canvas renderer*, introduciendo en él el panel a modificar, y se le ha añadido los componentes de *Image, Mask* y *Animator*, este último para animar el cambio de posición.

El componente *Image* define un espacio de ventana correspondiente al tamaño del objeto que lo contiene y *Mask* hace que todo lo que esté fuera de ella sea recortado, como se puede ver en la Ilustración 28 este componente había sido deshabilitado para poder ver la interfaz completa ya que al ser activado el aspecto que adquiere los aspectos que se pueden ver en la Ilustración 21 y la Ilustración 22.

Este mismo procedimiento se ha seguido para generar los tres paneles inferiores de sonidos, efectos de video y audio pero añadiéndo una barra deslizadora *(Scrollbar)* para poder desplazarse por dichos paneles de forma manual.

Ilustración 29. Estructura menú dinámico

Para relacionar la barra deslizadora y el panel que tiene que verse afectado por ella se ha de añadir un componente denominado *Scroll Rect*. Una vez añadido al objeto creado para la *ScrollViewk,* en este caso *MenuDinamico,* en la ventana de propiedades junto a *Image* y *Mask* aparecerán las opciones de este componente.

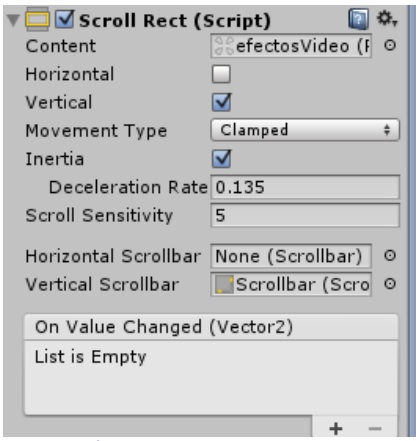


Ilustración 30. Componente menú dinámico

Para que funcione habrá que configurarlo añadiendo en *Content* el *Rect Transform* del panel que se debe de ver afectado por la barra deslizadora, la cual también habrá que añadir en *Vertical Scrollbar* en el caso de que sea vertical, para ello bastara con arrastrar los objetos que los contienen desde la ventana de *Hyerarchy* a los lugares previamente mencionados.

Con todo esto solo quedaría comentar los elementos que representan a los instrumentos pero estos son paneles con botones los cuales ya han sido comentados anteriormente, estos paneles se muestran según el instrumento que tenga seleccionado el interprete que está siendo configurado.

3.7. Funciones del servidor

El servidor es el que presenta la mayor carga de funcionalidad de este proyecto, ya que es el encargado de gestionar todos los intérpretes que se conectan, las distintas configuraciones de controles para cada uno de ellos, dar funcionalidad a la interfaz, interpretar los distintos mensajes recibidos... por ello en este apartado se explicarán de forma global las diferentes características de cada apartado y más tarde en el apartado 5 se explicarán los detalles de implementación.

Como se puede observar en la Ilustración 20 anteriormente mostrada, la aplicación se divide en cuatro ventanas, configuración del servidor, número de intérpretes, información interpretes y una gran ventana vacía que denominaremos 'ventana organizadora' cuando se haga referencia a ella, además esta ventana tiene dos posiciones como se ha visto anteriormente en el apartado 3.6 lo que conlleva una mayor carga de funcionalidades.

Configuración del servidor

Es usada para configurar el servidor con la información indicada y ponerse a la escucha de conexiones, además muestra la dirección IP que deben emplear los intérpretes para conectarse desde la aplicación móvil, en este caso '192.168.1.26'.

Una vez pulsado el botón de conectar este pasará a estar inactivo y cobrará un color verde.

Número de interpretes

Establece la cantidad de conexiones que se podrán establecer y crea tantos botones como filas por columnas se hayan establecido en la ventana organizadora calculando las posiciones que ocuparan los avatares de cada intérprete dependiendo del botón que los represente.

Al fijar el número de filas y columnas se crearán los respectivos botones, como se menciona anteriormente, y este botón junto a los *InputField* de edición se volverán inactivos, en la ventana de información de interpretes se habilitarán los botones de intercambio y reinicio.

Información de interpretes

Muestra la información disponible que haya sobre cada uno de los actuales interpretes. Está información se actualiza cada vez que se realice un *click* sobre uno de los botones presente en la ventana organizadora, mostrando la información disponible sobre el interprete al que representa.

El botón de intercambio será usado para cambiar a un interprete el botón de la ventana organizadora que lo representa, lo que conlleva un cambio de posición para el avatar de dicho interprete, a continuación se explicará el proceso ayudándose de ilustraciones para clarificar la explicación:

Para iniciar el intercambio una vez activado el botón, este se volverá de color verde y cambiará el efecto que provocará activar uno de los botones de la ventana organizadora dejando de mostrar la información al ser activado y realizando el siguiente proceso.

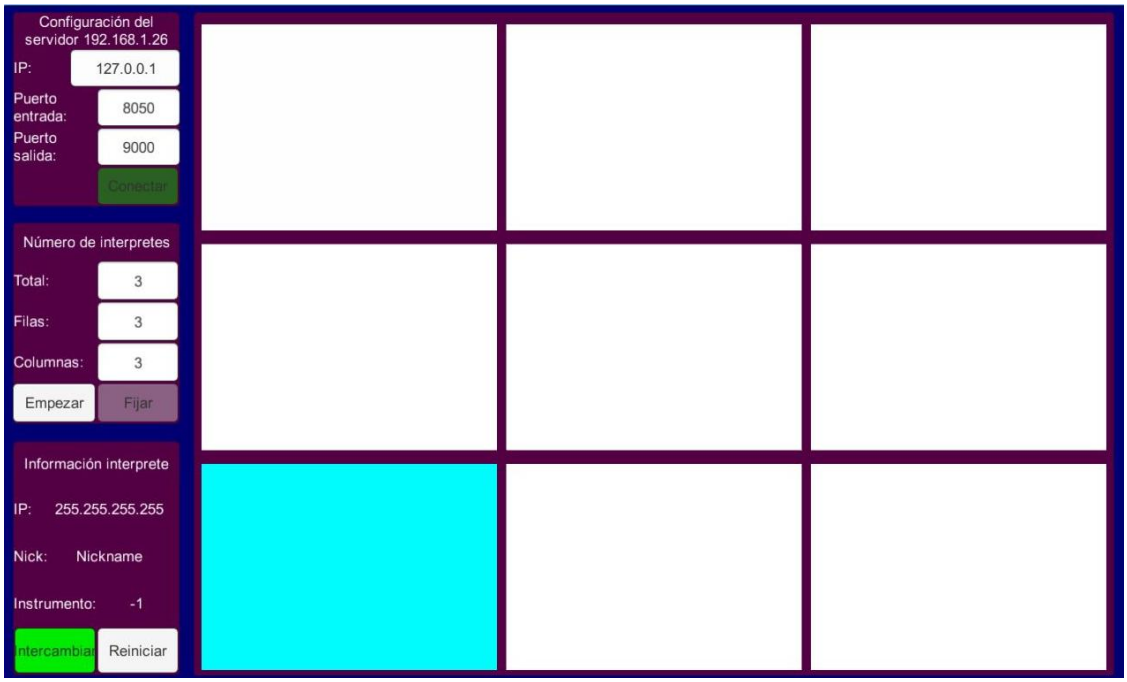


Ilustración 31. Intercambio inicio

El primer botón activado se volverá de color verde, a modo de indicar que ese será uno de los implicados en el cambio, que puede verse en la siguiente ilustración

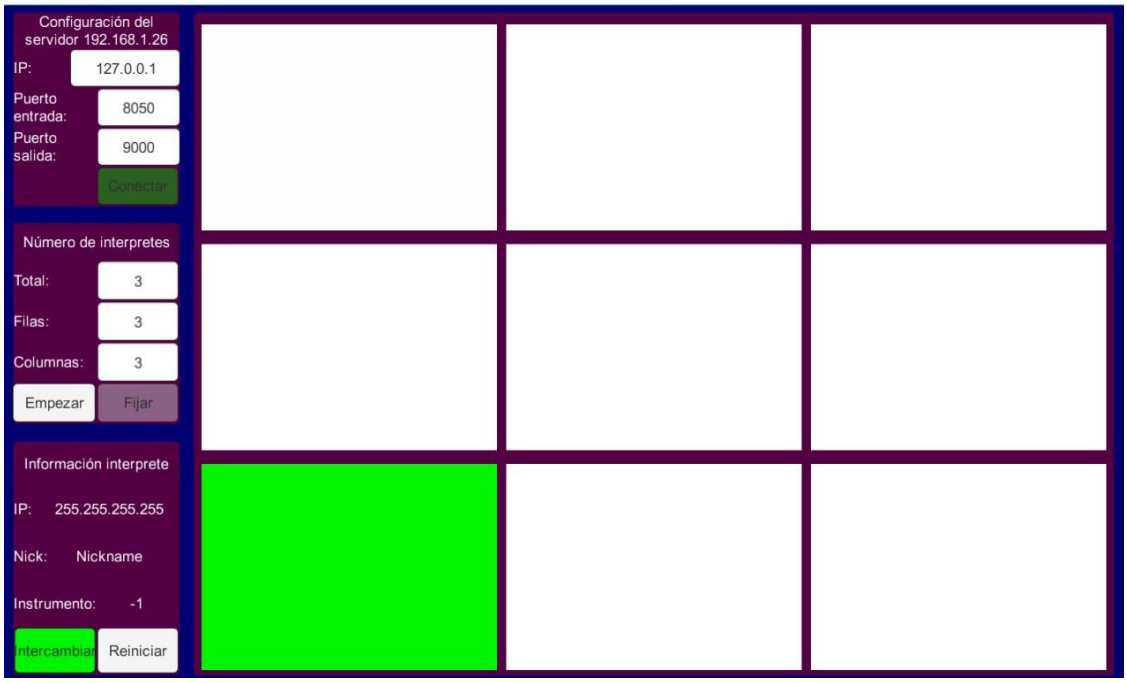


Ilustración 32. Intercambio

al seleccionar otro de los botones presentes el intercambio finalizará lo que retornará al color inicial a todos los botones, es decir, el botón de intercambio volverá a ser blanco, y los botones cuyos interpretes han sido intercambios cobrarán el color correspondiente a su estado actual, es decir, si al finalizar el intercambio se encuentran representando a un intérprete estos se volverán de color *cyan*, en caso de que al intercambiar su referencia a intérpretes sea nula cobrará un color blanco.

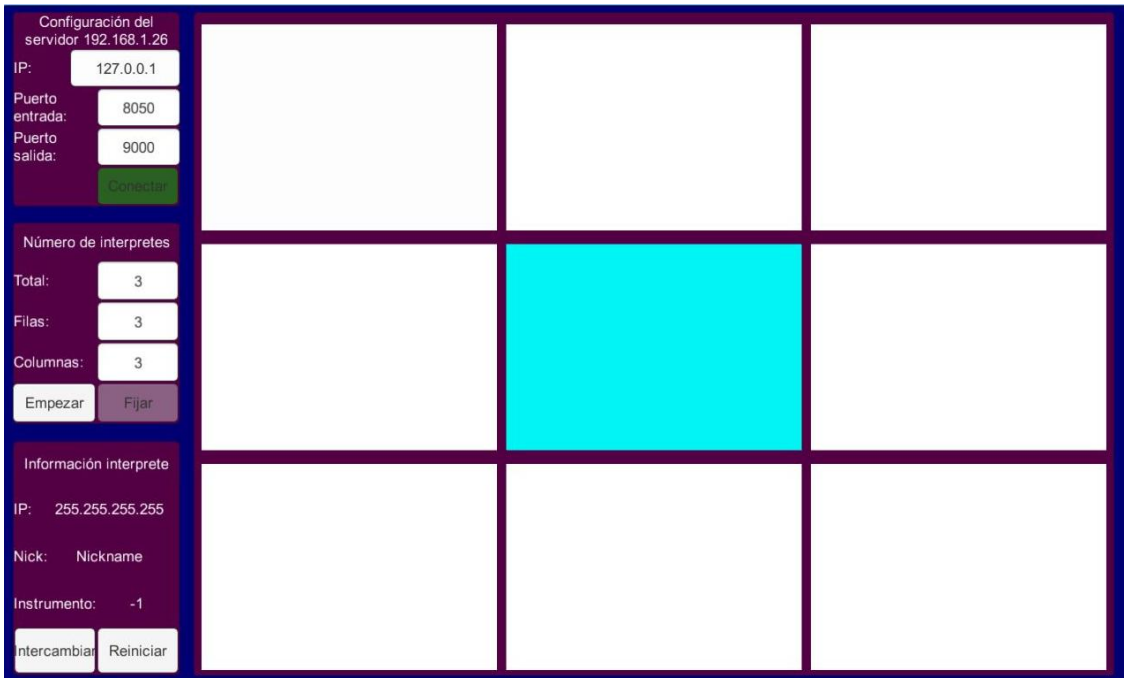


Ilustración 33. Intercambio final

Al terminar el intercambio los intérpretes a los que representan cada botón serán intercambiados, si solo uno de ellos tenía un intérprete se hará un cambio de representante por lo que uno de ellos seguirá vacio.

Este proceso de intercambio puede ser cancelado en cualquier momento volviendo a pulsar sobre el botón 'Intercambiar' lo que restaurará el estado inicial antes del inicio.

El otro botón que se encuentra en este panel es el de reiniciar el cual al ser pulsado deshace todos los posibles intercambios realizados y coloca a cada intérprete en la posicione que le corresponde por orden de llegada, por ejemplo si se han insertado cinco interpretes e intercambiado posiciones para conseguir la siguiente disposición

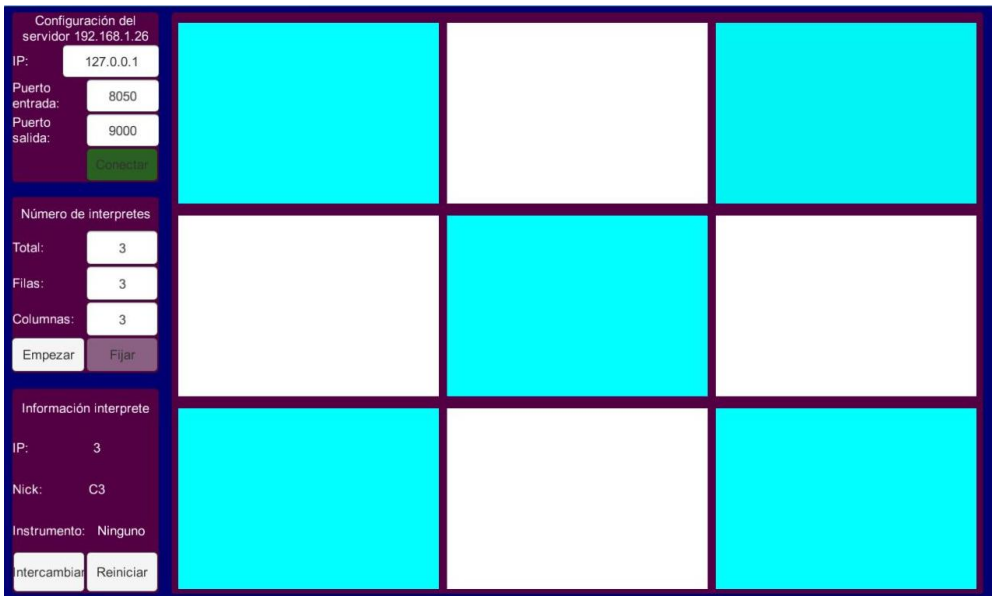


Ilustración 34. Ejemplo reiniciar

al pulsar el botón de reiniciar se reorganizará por orden de llegada quedando la situación que correspondería a no haber realizado ningún intercambio

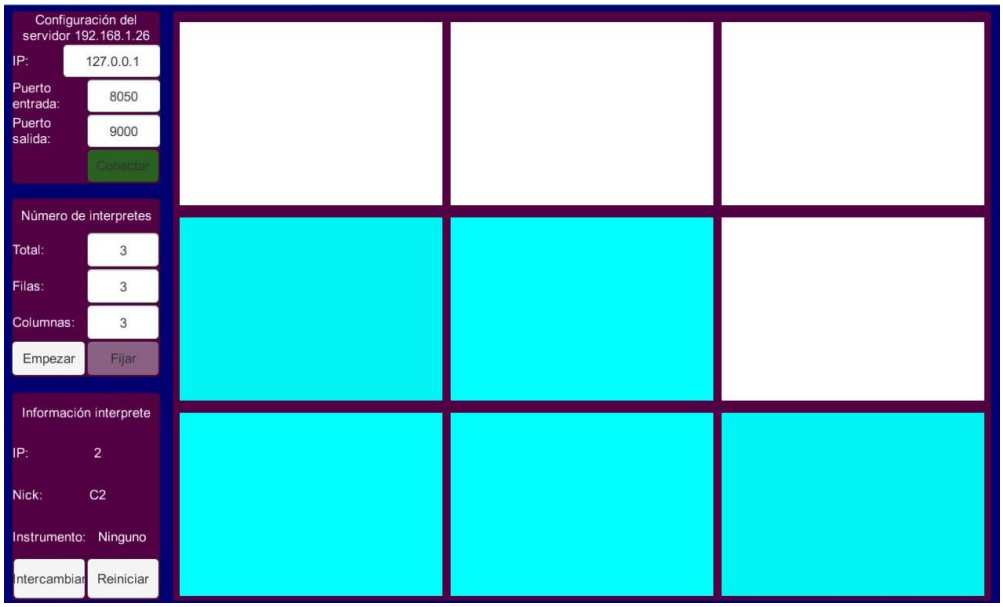


Ilustración 35. Ejemplo reiniciado

Ventana organizadora

En ella se mostrarán los botones que representarán las plazas disponibles para interpretes, al pulsar sobre un botón de intérprete con instrumento seleccionado, esta ventana cobrará el aspecto presente en la Ilustración 22 en la cual se observan cuatro paneles que son: instrumento, sonido, efectos de video y efectos de audio.

Instrumento representa a lo que está visualizando el interprete, al pulsar sobre uno de los botones de este panel , el cual pasará a ser de color verde, se cargarán los posibles efectos previamente asignados en el, esto se mostrará coloreando de verde aquellos botones de los distintos paneles de sonidos, efectos de video y efectos de audio.

Las opciones del menú 'Sonidos' genera distintos audios dependiendo de la opción seleccionada, los de 'Efectos de video' afectan al avatar de dicho interprete realizando distintas modificaciones visuales y las de 'Efectos de audio' son de asignación única para todos los intérpretes, es decir, no habrá una opción de dicho panel controlable por varios intérpretes o varios controles, a diferencia de los otros paneles que pueden repetirse entre interpretes e incluso en el mismo.

Todos los efectos pueden ser asignados y desasignados formando un mapeado entre los controles de los instrumentos y los distintos efectos posibles. Para controlar la asignación única del panel de 'Efectos de audio' estos solo podrán ser reasignados a otro control o interprete si se han desactivado del anterior controlador. Aquellos efectos de audio asignados a un control no serán accesibles para el resto ya que aparecerán como deshabilitados.

Ilustración 36. Mapeado de efectos

En esta imagen se puede observar como ciertos efectos de audio están inaccesibles lo que indica que han sido asignados a otro controlador o interprete.

3.8. Diagrama de clases

En los siguientes esquemas se podrán ver los diferentes scripts creados y la relación existente entre ellos, facilitando así posibles modificaciones futuras.

Cliente

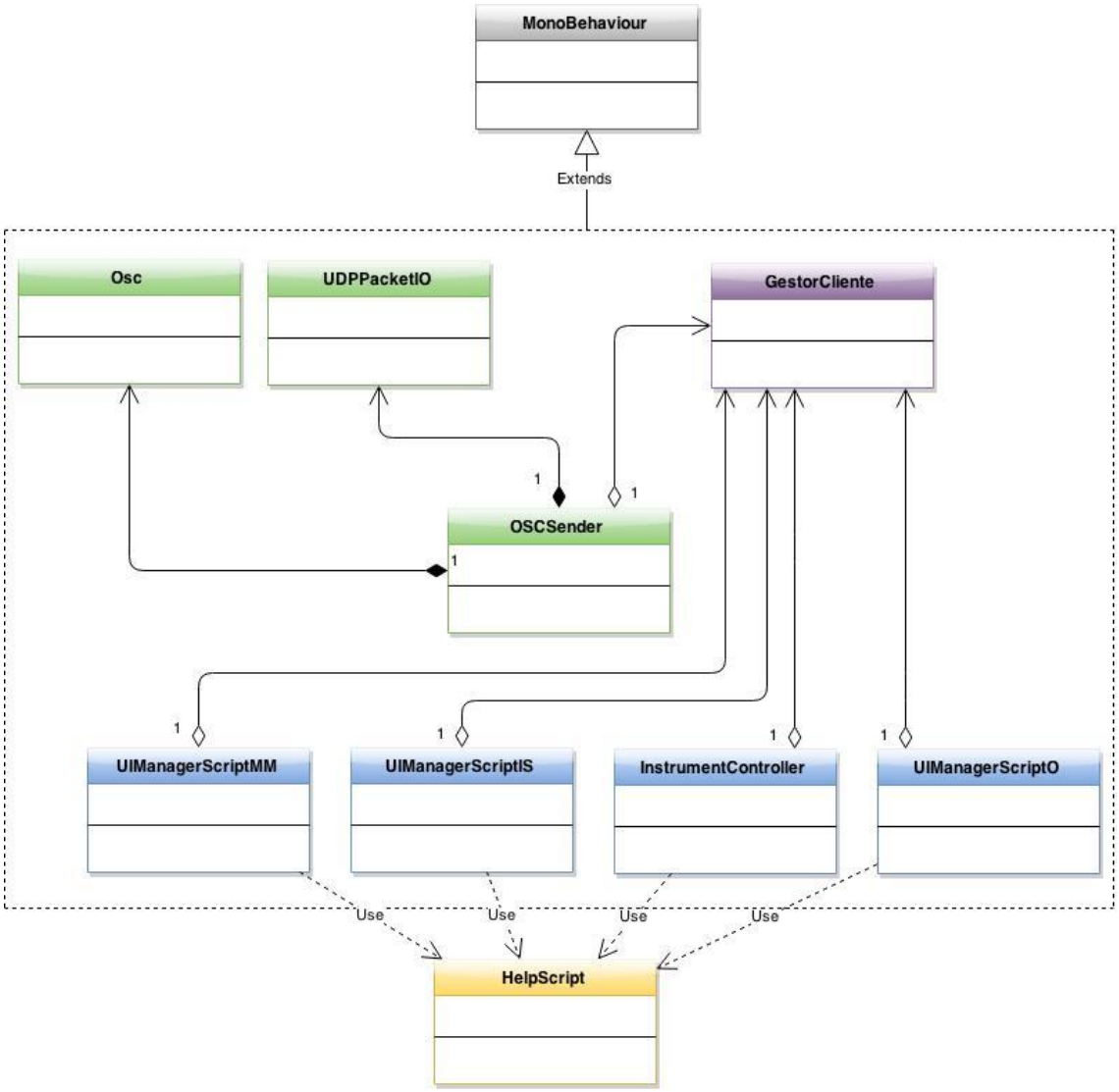


Ilustración 37. Diagrama de clases - Cliente

La clase de color gris es una de las proporcionadas por *Unity3D*, de la cual extienden las clases del interior del recuadro, es la clase básica para el manejo de los *GameObjects* y suele extender de todos los *scripts* salvo algunas excepciones, como es en este caso *HelpScript*.

La clase *HelpScript,* de color amarillo, contiene los métodos creados para agilizar la codificación del resto de *scripts* reduciendo complejidad y extensión.

Las coloreadas de verdes son las encargadas de gestionar las comunicaciones usando el protocolo *OSC*.

Las azules gestionan las interfaces, el control de los botones durante las diferentes ventanas de la aplicación, de ahí las terminaciones de sus nombres 'MM' (*MainMenu*),

'IS' (*InstrumentSelection*) y O (*Options*), para el conjunto de instrumentos se ha conseguido reducir a un único *script* el cual se denomina *InstrumentController*.

Por último está la clase morada, que hace las laboras de persistencia dentro de la aplicación.

Servidor

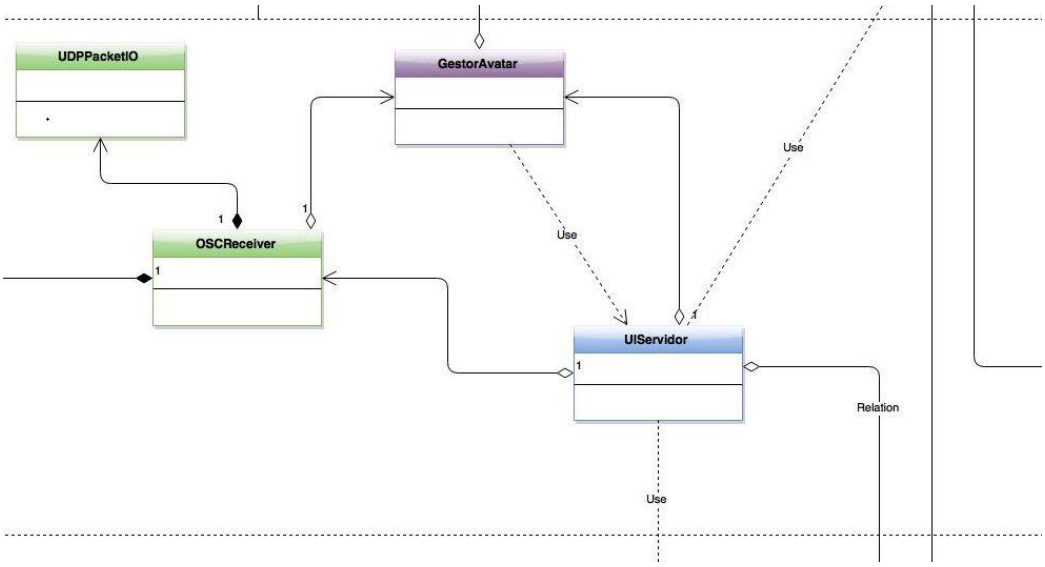


Ilustración 38. Diagrama de clases - Servidor

En el anexo I se puede ver el diagrama completo donde se puede observar que la mayor complejidad del proyecto reside en la aplicación servidora ya que es la encargada de:

- Identificar y gestionar a los intérpretes
- Programar los distintos efectos disponibles
- Permitir la configuración individual de cada uno de los interpretes
- Gestionar una interfaz que facilite la labor del usuario
- Reproducir los efectos de video y audio
- Gestionar los efectos únicos de audio

Para todo ello se han generado las clases mostradas en el anterior diagrama teniendo siete grupos diferenciados por los siguientes colores:

- Verde: encargado de gestionar las comunicaciones
- Morado: persistencia y lógica de aplicación
- Azul: gestiona la interfaz otorgando funcionalidad a los diferentes botones y comunicando los diferentes *scripts* de persistencia, lógica y comunicaciones.
- Rojo: *scripts* generador de sonidos y efectos de audio y video
- Naranja: interfaces que aplican los diferentes *scripts* de generación de audio y efectos.
- Amarillo: contiene los métodos creados para agilizar la codificación del resto de *scripts* reduciendo complejidad y extensión.

4. Organización

En este apartado se mostrará la organización usada para el control de los diferentes elementos usados en ambas aplicaciones, hay que tener en cuenta que todos los recursos usados en una aplicación desarrollada en *Unity3D* estarán siempre dentro de una carpeta llamada *Assets*.

4.1. Cliente

Ilustración 39. Organización cliente

Para la aplicación del cliente la estructura es muy simple, teniendo una carpeta para guardar las distintas escenas y otra para los *scripts* mostrados en el diagrama de clases, en los que los relacionados con comunicaciones se guardan en *OSC* y *plugin*, los relacionados con interfaces en *UIManager* y el resto, persistencia y general dentro de la carpeta general de *Scripts*.

4.2. Servidor

Ilustración 40. Organización Servidor

Para el servidor ha sido necesario hacer uso de una nueva carpeta requerida por *Unity* llamada *Resources* para la carga de elementos mediante *scripts*.

Además al ser la parte que mayor peso tiene del proyecto hace necesario del uso de diversas carpetas para conservar un orden coherente y que facilite la búsqueda de recursos.

Se hace uso de una carpeta de animaciones en la cual se guardan aquellas que realizan el cambio de posiciones en la ventana organizadora, pasando del estado mostrado en la Ilustración 21Ilustración 20 al de configuración de interprete mostrado en la Ilustración 22 y viceversa.

En la carpeta de *Prefabs* se guardan los elementos que se han diseñado previamente y usado más tarde, para realizar copia de forma rápida, como es el caso de los menús con barra deslizadora (*scrollbar)* o la figura 3D que representa a cada intérprete.

Por último se encuentra la carpeta de *scripts* en la cual se pueden observar que contiene otras carpetas para organizar correctamente los distintos tipos de scripts presentes dependiendo de la función que desempeñan, dejando la siguientes carpetas:

- Comunicación: gestionar las comunicaciones de red
- Efectos: generar sonidos y manejar efectos de video y audio
- Estructuras: *scripts* que definen diversas estructuras internas usadas
- Interfaces: interfaces implementadas por los *scripts* de Efectos
- ScriptsGenerales: *script* con código independiente del caso de uso

5. Implementación

En este apartado se expondrán los detalles de implementación que dan la funcionalidad a ambas aplicaciones desarrolladas, la usada por los clientes y el servidor.

5.1. Cliente

Para explicar los diferentes *scripts* que incluye se comenzará por describir la funcionalidades generadas en el *GestorCliente* que es el de mayor de la aplicación para intérpretes, como se puede observar en la Ilustración 37. Diagrama de clases - Cliente donde se observa su mayor número de enlaces y, una vez concluida, se pasará a explicar el *script* de utilidades usado para facilitar la implementación del resto para poder detallar con las bases necesarias los *scripts* de comunicación e interfaces.

GestorCliente

Es el encargado de la persistencia entre las diferentes escenas. En ella se guarda la información que debe mantenerse durante la ejecución de la aplicación como son el *nick* del usuario, el instrumento seleccionado, la dirección *IP* destino y los puertos de salida y entrada, incluyendo los respectivos métodos para guardar dicha información.

Se ha realizado el *script* para que sea del tipo *singleton*, es decir, que solo exista una instancia de este, ya que los datos generados son únicos por cada intérprete. Para ello se ha implementado un método que funciona como sigue:

- Al ser llamado, si no existe ninguna instancia previa, crea un nuevo objeto
- Se indica que no sea destruido al cargar una nueva escena
- Se instancia y añade el *script* como componente de ese nuevo objeto
- Añade los *scripts Osc* y *UDPPacketIO* que serán necesarios posteriormente para las comunicaciones
- Devuelve la instancia creada, los pasos anteriores a este solo se realizan si se cumple la condición inicial.

El código se puede ver a continuación

```
public static GestorCliente getInstance()
{
    if (manager == null)
    {
        GameObject MasterManager = new GameObject("MasterUIManager");
        DontDestroyOnLoad(MasterManager);
        manager = MasterManager.AddComponent<GestorCliente>();
        MasterManager.AddComponent<Osc>();
        MasterManager.AddComponent<UDPPacketIO>();
    }
    return manager;
}
```

Ilustración 41. Código singleton

esta es la estructura que siguen aquellos *script* del tipo *singleton* durante todo el proyecto.

Siguiendo la explicación de los métodos se encuentran los *getters and setters* de las distintas variables, tras ellos están aquellos empleados para la configuración de red y envió de mensajes que se ven a continuación.

La función *setRoute(string name, string dirIP, int sendPort, int listenerPort)* configura las distintas variables en función a lo introducido por el usuario en el menú de opciones, además envía un mensaje informando al servidor de la intención de conectarse a él, lo que provocará una reacción en este que se puede ver en el apartado o.

El método *setInstrumentSelected(int selected)* establece el instrumento seleccionado una vez es seleccionado desde el menú de selección de instrumento por el usuario lo que provocará el envió de un mensaje informando al servidor.

Por último el método *play(string hit, float value)* que es llamado cada vez que se modifica un *scrollbar* o se pulsa un botón de los paneles de instrumentos lo que envía un mensaje al servidor con la *IP* del intérprete, un identificador del botón o *scrollbar* modificado y el valor que este tiene, la estructura de los mensajes se puede ver más detalladamente en el apartado o.

HelpScript

La empleada en el cliente es una versión muy reducida de este *script* ya que solo son necesarios dos de los métodos empleados en ella *T getSingleUI<T> (string name)* y *T[] getListUI<T>(string tag)* son dos formas rápidas de obtener los componente de los objetos, conociendo el nombre de estos o un lista por la etiqueta. Son métodos genéricos por lo que el tipo de componente a obtener es indiferente, en este caso serán componente usados en la interfaz como botones y *scrollbar*s.

Comunicación

Los *scripts Osc* y *UDPPacketIO* son los usados por *OSCSender* y que fueron implementados por (Heavers). Para esta aplicación se ha modificado el *OSCSender* para el cual se han introducido pequeños cambios, como obtener la información de conexión de *GestorCliente* llamando a las respectivas funciones, crear un método para enviar mensajes a la dirección IP establecida y otro con un mensaje final de desconexión al ser deshabilitado el *script*.

Interfaces

Hay diferentes *scripts* para el control de las ventanas tres para el menú principal, opciones y selección de instrumento y otro para los tres paneles de instrumentos.

Sus funciones se basan en recoger la información y acciones proporcionadas por el usuario, con la ayuda de las funciones de *HelpScript*, y pasarlas al *GestorCliente* para que las manipule dependiendo de lo explicado anteriormente, ya sea guardando la información o generando mensajes para el servidor.

El menú principal está configurado para salir de la aplicación, cargar el menú de selección de instrumento si previamente se han configurado las opciones y cargar el menú de opciones.

El *script* de opciones va guardando los diferentes valores conforme el usuario los modifica y una vez pulsa el botón de guardar este llama a la función *setRoute* del gestor de cliente con los datos introducidos para establecer la conexión.

El de selección de instrumento carga los diferentes paneles, dependiendo de la tecla pulsada y llama a la función *setInstrumenSelected* del gestor para enviar un mensaje informando al servidor.

Para acabar está el que controla los diferentes paneles de instrumentos, el cual añade a cada botón y barra deslizadora una función que, al ser pulsado o cambiado su estado respectivamente, llama a la función *play* para generar mensajes que informen al servidor del controlador pulsado o modificado y el valor que este tiene.

5.2. Servidor

Para explicar la parte del servidor se seguirá el siguiente orden *ScriptsGenerales*, *Estructuras*, *GestorAvatar*, *Comunicacion*, *UIServidor*, *Interfaces* y *Efectos*.

***Scripts* Generales**

En el servidor se encuentra la versión completa del *HelpScript*, que además de los métodos mencionados anteriormente en el apartado 0 incluye estos:

Métodos de obtención de información

- *string[] getMethods<T>():* es usada para obtener los nombres de métodos de alguna clase o interfaz. Se emplea para obtener los nombres de los métodos en las interfaces implementadas para la generación de los menús dinamicos.

- *logMethods(string[] metodos)*: función de para *debuggear* que imprime por consola los nombres de todos los métodos encontrados.

Creación de elementos de interfaz

- *GameObject nuevoCanvas(string name):* función creada para facilitar la creación de *Canvas* que no terminó siendo usada.

- *GameObject newPanel(string name, GameObject padre):* realizada para facilitar la creación de elementos de interfaz que no terminó siendo usada.

- *GameObject newButton(string name, GameObject padre, float minX, float minY, float maxX, float maxY)*: método que simplifica la creación de botones para la interfaz y es usada tanto para crear los botones de la ventana organizadora (véase Ilustración 21. Ejemplo organización 3x3) como en un método de este mismo *script* denominada *menuDinamico*

  Este método está sobrecargado para poder indicar, además de las coordenadas, el espaciado con los bordes *GameObject newButton(string name, GameObject padre, float minX, float minY, float maxX, float maxY, float left, float bottom, float right, float top)*

- *GameObject newText(string name, GameObject padre,float minX, float minY, float maxX, float maxY,string msg)*: Forma de añadir textos a los diferentes elementos de la interfaz. Usada para añadir texto en los botones creados que lo requieran como son los pertenecientes a los paneles de efectos y sonidos.

- *GameObject newInputField(string name, GameObject padre,float minX, float minY, float maxX, float maxY,string msg)*: Forma de crear cajas de inserción de texto posteriormente no usadas en el proyecto.

- *GameObject[] menuDinamico(GameObject padre, string[] nombres)*: al pasarle como padre un objeto del tipo panel y una lista de nombres, esta función crea un botón por cada uno de esos nombres dentro del panel con su respectivo texto. Es usado para las paneles de sonido, efectos de audio y efectos de video.

Modificación de elementos de interfaz

- *cambiaColorBoton(GameObject boton, Color color)*: dado el objeto que contiene un botón cambia su color actual por el pasado como parámetro.

  Función sobrecargado con una segunda posible llamada donde se indica el nombre del objeto *cambiaColorBoton(string boton, Color color)*

  Estructuras

Para el desarrollo y control de las diferentes partes de este proyecto ha sido necesario establecer unas estructuras de datos más tarde usadas por los distintos *scripts* presentes en la aplicación. Estas son *AvatarInterprete,* que a su vez contiene una estructura interna denominada *Efecto,* y *Organizador*.

Efecto

Estructura formado por tres *string* cada una de ellas para determinar qué efecto está asignado en generación de audio, efectos de audio y efectos de video para cada botón de los intérpretes, esta estructura tiene un método de instanciación y sus respectivos *getters and setters*.

AvatarInterprete

Estructura empleada para guardar la información que hace referencia a cada uno de los intérpretes que se conecten al servidor. Estos datos que son usados durante la aplicación son los siguientes:

- ID: identificador, en este caso la *IP* del dispositivo

- Nombre: introducido por el usuario en las opciones

- Objeto: que hace de avatar de dicho interprete

- Instrumento: el panel de control seleccionado

- Efectos: diccionario del tipo *Dictionary<string, Efecto>* donde la clave es una cadena de caracteres que hacer referencia a uno de los botones del panel y el contenido es una estructura de *Efecto* en la cual se guarda las asignaciones realizadas a dicho botón

- Controlador de video: instancia de la clase *ControladorVideo* empleada para la manipulación de efectos visuales

- Controlador de generación de audio: instancia de la clase *ControladorGeneradorAudio* que reproduce los sonidos implementados en dicha clase a través de su propia fuente.

- Controlador de efectos de audio: puntero al *singleton* de la clase *ControladorEfectoAudio* que permite la modificación de los efectos que repercuten sobre el resultado final de los sonidos

- Fuente de sonido: sobre la que se reproducen los sonidos y cuya salida está conectada al mezclador general de audios.

Además de los respectivos métodos para obtener los valores y modificarlos de las respectivas variables está el método *play (string boton, string valor)* que llama a las funciones de las clases de generación de sonidos y efectos de audio y video según lo asignado por el usuario del servidor.

Organizador

Estructura destinada a mantener la relación entre los botones de la ventana organizadora y los intérpretes creados, en ella se guardan las siguientes referencias:

- Objeto del botón: *GameObject* al objeto que contiene el botón

- Boton: *Button* al componente botón del objeto

- Intérprete: al intérprete correspondiente al actual botón

Al ser instanciada está estructura calcula la posición que debería de ocupar el avatar al que hacer referencia y lo guarda en una variable de tipo *Vertor3*.

Además de los tradicionales métodos para manipular las variables privadas están los siguientes:

- *selecionado():* cambia el color del botón a uno establecido como estado seleccionado

- *posicionar():* si la referencia interprete es distinto de nulo se posiciona su avatar a las posiciones correspondientes al actual botón.

- *reiniciar():* elimina las referencias al intérprete y restaura el color del botón a vacio

- *reiniciaColor():* si la referencia a interprete es distinto de nulo se establece el color del botón al del estado lleno, en caso contrario su color se establece al del estado vacío

GestorAvatar

Es un *script* del tipo *singleton* diseñado para controlar el número de interpretes que se conectan al servidor, para ello se han declarado las siguientes variables:

- Interpretes: del tipo *Dictionary<string, AvatarInterprete>* en él se guardan diferentes estructuras del tipo *AvatarInterprete* teniendo como clave una cadena de caracteres, en este caso la *IP* del dispositivo para asegurar la creación única de un intérprete.

- Número de interpretes: representa al número actual de interpretes que se han unido al servidor

- Máximo de interpretes: indica el número máximo posible de interpretes a crear, el cual es establecido desde la interfaz del servidor.

También incluye los siguientes métodos:

- *newAvatar(string id, string nick)* : Es usado para agregar nuevos interpretes, siempre y cuando no exista el identificador actual y no exceda el número máximo de intérpretes. Instancia un nuevo avatar e intérprete, el cual es agregado al diccionario para controlarlo junto al resto, por último se incrementa la variable del número de interpretes y se llama a la función *nuevoInterprete(AvatarInterprete interprete)* del *script UIServidor* el cual se explica en el apartado 0

- *getInterprete (string id)* : devuelve, en el caso de existir, el interprete de identificador *id,* en caso contrario *null*

- *existeAvatar(string id)* : busca si existe algún interprete con dicho identificador y devuelve un *bool*

- *setInstrument(string id, int instrument)* : establece el instrumento al intérprete de identificador *id*

- *setMaxAvatar(int max)* : establece el número máximo de interpretes posibles para la actual sesión

- *playAvatar(ArrayList mensaje)* : método que extrae los valores del mensaje para llamar más tarde a la función *play* de la estructura *AvatarInterprete*, vease en apartado 0

```
public void playAvatar(ArrayList mensaje)
{
    string id = mensaje [0].ToString ();
    string boton = mensaje [1].ToString ();
    string valor = mensaje [2].ToString ();
    interpretes [id].play (boton, valor);
}
```

Ilustración 42. Función playAvatar

Comunicación

En comunicación al igual que en el caso del cliente tenemos los *scripts Osc* y *UDPPacketIO* que son usados, en este caso, por *OSCReceiver* y que fueron implementados por (Heavers). Para esta aplicación se ha modificado el *OSCReceiver* para el cual se han realizado los siguientes cambios.

Para empezar se ha definido una estructura interna denominada *Nuevo* la cual es empleada para llevar un control de los intérpretes que ya se han intentado crear. Esto se debe a que Unity3D nos obliga a realizar la llamada de nuevas instancias desde una de las funciones *Start, Update* o *FixedUpdate*.

En esta estructura solamente se guardan una cadena de caracteres que identifican al intérprete y una *booleano* que indica si ha sido ya llamado a instanciar desde la función *Update*.

Una vez mencionada la estructura interna y su uso se procederá a explicar el resto de cambios introducidos al código original que son los siguientes.

Se ha declarado un diccionario del tipo *Dictionary<string, Nuevo>* para controlar las diferentes peticiones de nuevos interpretes, además se ha añadido el método *conectar(string ip, int puertoEntrada, int puertoSalida)* el cual será llamado desde el script *UIServidor* para establecer la conexión con los datos proporcionados por el usuario mediante la interfaz.

Por último se ha modificado el método *AllMessageHandler(OscMessage oscMessage)* para que contemple los siguientes posibles mensajes:

- */new ip name* : al recibir un mensaje de este tipo añadirá al diccionario anteriormente mencionado una nueva entrada
- */instrumento ip value* : llama a la función *setInstrumento* de *GestorAvatar* para establecer el instrumento del interprete
- */play ip boton value* : llama a la función *playAvatar* de *GestorAvatar*

Tanto en */instrumento* como en */play* se comprueba previamente si se ha introducido con anterioridad dicho interprete.

UIServidor

Como se puede ver Ilustración 38. Diagrama de clases - Servidor este es el *script* que más enlaces tienes con el resto, lo que termina indicando que es el que mayor complejidad presenta.

Es el encargado de dar funcionalidad a la interfaz presentada al usuario y para ello hace uso de muchos de los métodos explicados con anterioridad, por lo que de forma similar a la anterior se expondrán los principales atributos declarados y explicará el trabajo que realiza cada una de las funciones.

En esté *script* se encuentran una gran cantidad de atributos, la mayoría usados para tener referencias a aquellas partes gráficas que se necesitará acceder para ser modificadas, como pueden ser paneles y botones entre otros, además de ellos están los siguientes:

- Lista de colores: usada para tener un *standard* de los estados de color de botones (normal, seleccionado, lleno,error...).

- Variables de control de cámara: distintas variables para controlar la velocidad de desplazamiento, dirección y zoom de la cámara de escena.

- Variables de comunicación: usadas para poner el servidor en escucha.

- Diccionarios del tipo *Dictionary <string, Organizador>*, *Dictionary <string, Button>* y *Dictionary <string, bool>* el primero de ellos usado para gestionar las relaciones entre los botones de la ventana organizadora y los intérpretes, el segundo guarda una relación entre el nombre de los efectos y los botones que los representan en los distintos paneles y, el último de ellos, es el que se encarga de que los efectos de audio sean asignables un única vez mediante un par nombre del efecto - *bool*.

- Animador: que sirve para controlar el momento en el que lanzar la animación empleada para controlar el *scroll* de la ventana organizadora.

- Variables para controlar distancias: existen dos variables destinadas a esto, una de ellas es la distancia entre los botones de la ventana organizadora, y la otra la distancia entre avatares.

- Variables intercambio: existe una variable usada para controlar el estado en el que se encuentra el proceso de intercambio el cual se explicará posteriormente y otra que indica si el proceso de intercambio está o no activo, además de guardar una referencia a los botones que serán afectados.

- Variables de configuración: por último están las variables que guardan el interprete que ha sido seleccionado para configurar y el último botón pulsado en el panel de instrumentos para configurar.

Tareas de la función *Start()*

En ella se inicializan todas aquellas variables que hacen referencia a los objetos de la interfaz además de asignarles a los botones aquellas funciones que deben activarse al hacer *click que* se pueden observar en la siguiente tabla:

Botón	**Función**
Conectar	*oscReceiverUp*
Fijar	*disposicionInterpretes*
Reiniciar	*reiniciaInterpretes*
Intercambiar	*swapObjects*
Efectos de video	*selectedVideoEffect*
Efectos de audio	*selectedAudioEffect*
Generadores de audio	*selectedAudioGen*
Panel instrumental	*selecInstrument*
Guardar	*finalizaConfiguracion*
Empezar	*startAvatar*

Tabla 2. Relación botón función

Además de ello se obtiene una referencia a la instancia *singleton* de *GestorAvatar*, obtiene el *script* de comunicaciones para ser configurado e inicializa la lista de colores para los diferentes estados.

Como último se emplea la función de *menuDinamico* y *getMethods* de *HelpScript* (vease oScripts **Generales**) para crear los diferentes botones de los tres paneles de los menús sonido, efectos de video y efectos de audio.

Funciones implementadas

A continuación se expondrán los métodos implementados y la funcionalidad que ofrecen cada uno de ellos:

- *oscReceiverUp()*: usado para poner al servidor en escucha pasando la información proporcionada por el usuario a la función *conectar* de *OSCReceiver*, además una vez está conectado deshabilita el botón de conectar y para a estar de color verde.

- *startAvatar()*: una vez se pulsa el botón de *Empezar* se oculta el panel de opciones y se dejan ver los avatares.

- *disposicionIntrepretes()*: Lee los datos de numero de interpretes maximo, numero de filas y columnas, comprueba que sean datos validos y, una vez lo son, se llama a las funciones *setMaxAvatar* de *GestorAvatar* y a la función *organizar* explicada a continuación, además deshabilita la configuración de los datos introducidos y el botón fijar a la vez que habilita los de intercambio y reiniciar.

- *organizar(int filas, int columnas)* : Crea los botones que representan a los intérpretes en las opciones de la aplicación, para ello calcula el tamaño de los botones dependiendo del espacio disponible y se va agregando al diccionario organizador cada uno de ellos asignado la función *seleccionado* para que se active al ser *clickeados*. Para concluir se llama a la función *recalcularCamara*.

- *nuevoInterprete(AvatarInterprete interprete)* : Método llamado externamente para informar de la existencia de un nuevo avatar y relacionarlo con uno de los botones que permanece sin asignación, además posicionar al *avatar* que lo representa visualmente.

- *reiniciaInterpretes()*: Método para reiniciar las posiciones de todos los intérpretes a una formación básica, con los datos obtenidos desde el GestorAvatar llama a la función *reiniciaInterpretes* que se comenta a continuación.

- *reiniciaInterpretes(Dictionary<string, AvatarInterprete> interpretes)* : se recorren todas las claves del diccionario organizador llamando a la función *reiniciar* explicada con anterioridad, una vez realizado se recorren todos los intérpretes a los cuales se les reasigna un botón organizador.

- *swapObjects()*: Comienza o detienes el proceso de intercambio de intérpretes, para ello realiza lo siguiente:

  Si la variable que indica si el proceso está activo es *false* entonces lo cambia a *true,* lo que afectará en la función *seleccionado* que se verá a continuación, además de cambiar el color del botón a verde y se inicia la variable que controla el estado del proceso.

  Si esta variable de contro está a *true* y se ha seleccionado ya un botón, se reinicia este botón a su color inicial, se pone la variable de control a *false* y el botón intercambiar cobra de nuevo su color natural.

  Por último si no se han cumplido ninguna de las condiciones anteriores simplemente se desactiva la función de control y restablece el color del botón al natural.

- *seleccionado(string buttonID)* : está varia su funcionalidad según si se está realizando un intercambio o no.

  Si se está realizando un intercambio se entra en un *switch* que según el valor de la variable de estado del proceso puede realizar una de las siguientes tares, en caso de ser el estado inicial guarda el *ID* de identificador del botón pulsado, cambia su color llamando a *selecionado* y se cambia el valor de la variable al del siguiente estado.

  En el siguiente estado es donde se culmina el intercambio, los intérpretes son cambiados en el diccionario organizador, se posicionan los respectivos avatares y reinician los valores de variables de intercambio.

  En caso de no estar en proceso de intercambio se muestra en la ventana de información los datos disponibles del botón de interpretes seleccionado y además, si contiene un intérprete con instrumento ya seleccionado se activa la animación que proporciona acceso al panel de configuración de intérprete, observable en la Ilustración 36. Mapeado de efectos activando el panel instrumento que tenga seleccionado dicho interprete. En caso de que el interprete relacionado al botón sea *null* se muestra información *standard*.

- *recalcularCamara()*: se hace un cálculo aproximado de la posición que permite una visión global de los avatares posibles en función del número de filas y columnas

- *finalizaConfiguracion()*: es la que se ejecuta al pulsar el botón de guardado y se encarga de restablecer a color normal a los botones de panel de instrumento, generación de sonidos y efectos de video y audio del último botón que fue pulsado, los deshabilita y oculta el panel instrumento que usaba dicho interprete, además habilita el botón empezar y ejecuta la animación para retornar a la visión de la ventana organizadora inicial (Ilustración 21).

- *selecInstrument(string buttonName)* : es el método que se activa al pulsar un botón del panel instrumento representante al del intérprete, al ser activado comprueba si previamente había otro, en caso afirmativo restablece su color y el de los efectos que tenga asignados al normal, una vez comprobado su color pasa a ser verde y el de los generadores de audio o efectos asignados a este también indicando la relación existente entre ellos. De forma especial para los efectos de audio se comprueba si tiene alguno asignado, que de ser así se habilitaría para poder interactuar con él.

A continuación se explicarán los métodos que afectan a la selección de efectos de video, audio y generadores de audio.

En el caso de generación de audio (*selectedAudioGen(GameObject obj, Button boton)*) y efectos de video (*selectedVideoEffect(GameObject obj, Button boton)*) las funciones solo varían en las llamadas realizadas a *AvatarInterprete* usadas para establecer las distintas propiedades.

En un principio se comprueba si previamente se tenía asignado una propiedad previa, en caso de ser así se restablece el color normal a dicha propiedad, una vez realizado se comprueba si la selección es la misma que la que estaba asignada, si es la misma se desasigna esta del botón del intérprete en caso contrario se asigna el nuevo valor borrando el anterior.

En el caso de la selección de efecto de audio (*selectedAudioEffect(GameObject obj, Button boton)*) además de lo mencionado anteriormente hay que configurar el diccionario de efectos de audio estableciendo la propiedad del actual seleccionado a *true* si se está asignado o *false* en caso de que se libere.

Funcionalidades otorgadas a teclas

Este *script* tiene además la implementación de ciertas funcionalidades a diferentes teclas como son:

- M : se activa el panel de opciones
- Flechas de dirección: mueve la cámara en las direcciones de cada flecha
- +/- : acerca la cámara a los avatares o la aleja
- Esc : cierra la aplicación

Interfaces

Hay tres tipos de interfaces, una por cada tipo de propiedad que se puede agregar a los distintos botones de los paneles instrumentales que son

- *InterfaceEfectoAudio*
- *InterfaceGenAudio*
- *InterfaceVideo*

En ellos se declaran las cabeceras de todos los métodos de generación de audio y efectos que se vayan a establecer

Efectos

En esta carpeta se encuentra los tres *scripts* que implementan las interfaces anteriormente mencionadas. Como son *scripts* que podrían ser extremadamente extensos se explicará el funcionamiento general de estos, explicando la función *callFuction(string name, string data)* común en todos ellos.

Está función es usada para posibilitar de una manera sencilla la llamada a los distintos métodos presentes en las interfaces, para ello se transforma la cadena de caracteres *data* a un valor con decimales *float* el cual es usada para llamar a las funciones.

Dentro del método existe un *switch* que hay que ir completando añadiendo casos por cada método encabezado en la interfaz que implementa el *script*, para su funcionamiento correcto cada caso se nombrará exactamente igual que el encabezado descartando los atributos que le sean necesarios

Interfaz	*callFunction*

`public interface InterfaceVideo {`
`    void setRotateX(float value);`
	`public void callFunction(string name, string data)`
	`{`
	`    float number = float.Parse(data);`
	
	`    switch (name) {`
	`    case "setRotateX":`
	`        setRotateX(number);`
	`        break;`

Tabla 3. Inserción de nueva propiedad

6. Ampliaciones realizadas

En este apartado se presentan todos los cambios realizados sobre el prototipo una vez conseguidos los objetivos iniciales. Se han realizado cambios en sendas aplicaciones con la intención de mejorar la calidad de estas, para lo cual se ha realizado lo siguiente:

6.1. Barra de estado cliente

Se ha introducido una barra de estado que se muestra en los distintos paneles de instrumentos y la selección de estos, está servirá para visualizar rápidamente información que podría ser necesaria sin tener que acudir al panel de opciones.

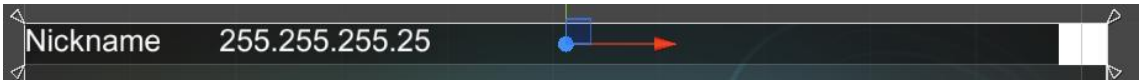


Ilustración 43. Barra de estado

En este elementos, colocado en la parte superior de la pantalla, se pueden observar tres campos distintos. El primero de ellos mostrará el nombre que ha introducido el usuario en las opciones de la aplicación, en el segundo se mostrará la *IP* del dispositivo y el último es una imagen la cual se ha insertado para una futura ampliación que servirá para indicar el estado de la conexión con el servidor.

Para implementar la funcionalidad de dicha barra se ha generado el siguiente método en el *script* de *GestorCliente*.

```
public void actualizarDatos()
{
    HelpScript.getSingleUI<Text> ("nickname").text = nick;
    HelpScript.getSingleUI<Text> ("direccionIP").text = my_ip;
}
```

Ilustración 44. Actualización de barra de estado

Hace uso de la clase HelpScript por lo que hay que añadir una nueva línea al diagrama de clases marcando la nueva relación existente que se puede observar en el anexo II. Este método es llamado desde las funciones *Start* de los *scripts UIManagerIS* e *InstrumentController* tras obtener la instancia de *GestorCliente*, lo cual actualiza la información mostrada en la barra de estado en cada escena.

```
private void Start()
{
    manager = GestorCliente.getInstance ();
    manager.actualizarDatos ();
```

Ilustración 45. Actualización de barra de estado

6.2. Icono cliente

Se ha cambiado el icono que se muestra por defecto en la aplicación de *Android* por el siguiente, accdeciendo a las opciones de configuración en *File -> Build Settings... -> Player Settings... -> Default Icon*

Ilustración 46. Icono Android

6.3. Mejoras visuales cliente

Se ha mejorado el aspecto visual de todas las ventanas del cliente cambiando las imágenes de los botones del menú inicial que los hace más intuitivos y eliminado sus texto, además se han modificado las imágenes de fondo los paneles y botones de todas las escenas, retocando la transparencia de los *scrollbar* para obtener un mejor impacto visual. En el anexo III se pueden observar las distintas ventanas con su nuevo aspecto.

6.4. Implementación de indicador de conexión

Se ha implementado la funcionalidad del indicador de conexión entre el cliente y el servidor de modo que aparece un circulo verde en caso de que esté conectado o uno rojo si no lo está.

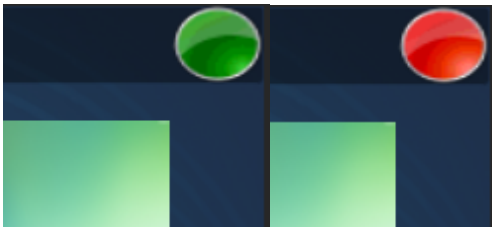


Ilustración 47. Estados del indicador

La forma de actuar para conocer si está conectado será la de enviar un mensaje periódicamente al servidor preguntando si se está conectado a lo cual éste tendrá que responder con un tipo de mensaje en concreto antes de un tiempo límite establecido.

Para ello ha habido que realizar modificaciones tanto en la aplicación del cliente como la del servidor.

En la parte del cliente se han modificados los *scripts GestorCliente* y *OSCSender*, en el primero de ellos se han generado dos nuevas funciones y variables además de modificar una tercera y en el segundo se ha introducido un manejador de mensajes denominado *AllMessageHandler* para indicar como actuar dependiendo de los mensajes recibidos, además en el mensaje que informa de la conexión de intérpretes se ha añadido un nuevo campo con la información del puerto a la escucha, que ha configurado el cliente, quedando de la siguiente manera */new ip puerto* el cual será usado desde el servidor para poder enviar mensajes.

La variables generadas son usadas para saber si se está conectado y si el servidor a contestado o no dentro del intervalo establecido.

La función modificada ha sido *actualizarDatos* haciendo que actualice el estado del indicador de conexión dependiendo del valor de la variable de control.

La primera de las funciones generadas es *conectadoON* cuya labor es la de establecer las variables de control añadidas a verdadero, indicando que se está conectado y se ha recibido respuesta ya que esta función es activada al recibir el mensaje del servidor.

La segunda es *sigoConectado* del tipo *IEnumerator* que se usa para comenzar las comprobaciones de conexión, está es iniciada al pulsar la primera vez sobre el botón de guardar en el panel de configuración de opciones y es la que se encarga de enviar los mensajes de comprobación de conexión al servidor cada cierto tiempo.

Tras enviar el mensaje se cambiará la variable de control de respuesta a falsa esperando a una respuesta para volverla verdadera, si no se recibido ninguna respuesta en el

tiempo límite se supondrá que se ha perdido la conexión y cambiará la variable de control de conexión a falsa y el icono al color rojo, en el caso de recibir el mensaje de confirmación se establece el color verde. Las comprobaciones seguirán realizándose periódicamente una vez iniciado el proceso, para ello se hacen llamadas recursivas de la función una vez concluye un ciclo de esta.

La última función añadida en el cliente es *AllMessageHandler*, en este caso en el *script* de *GestorCliente*, está será la encargada de decidir cómo actuar en función del mensaje recibido, para implementar la funcionalidad de esta característica se ha decidido un mensaje simple en el que se indica la dirección del mensaje dentro de la función manejadora, en este caso será "/conectado", al recibir este mensaje se llamará a la función anteriormente mencionada *conectadoON* para indicar que se ha recibido una contestación.

Además de estos cambios realizados en el cliente se han realizado modificaciones en el servidor. Se ha rediseñado la estructura de los intérpretes *AvatarInterprete* haciendo que incorporen nueva información necesaria para enviar mensajes a los clientes.

Se genera un objeto que contiene los *scripts* de comunicaciones e inicializa en el constructor de la clase con la información enviada desde el cliente que son la *IP* y puerto al que enviar los mensajes. Una vez solucionado el problema de la conexión se ha creado una nueva función *conectado* para responder, la cual genera y envía un mensaje con la dirección "*/conectado"* desde el intérprete que es llamada.

Esta información ha sido añadida a la estructura *AvatarInterprete* ya que es algo que deberá hacer cada uno de ellos y son controlados por *GestorAvatar,* en el cual se ha actualizado la llamada al constructor de la estructura intérprete para agregar los puertos a los que enviar información y por los que escuchar en dicha conexión, este último se generará automáticamente a partir del puerto de escucha del servidor sumando una unidad por cada nuevo interprete. Una vez se llama a la función *newAvatar* de este *script* y se ha generado el nuevo interprete se llama a la función *conectado* del recientemente creado para informar de que se ha establecido la información con éxito.

Además se ha generado la función *sigueConectado* la cual es llamada desde el *script OSCReceiver*, esta función busca en el diccionario de intérpretes al que ha solicitado la confirmación de conexión y llama a su función *conectado* para generar el mensaje informándole. Para ello se ha añadido un nuevo caso a la función *AllMessageHandler* del *OSCReceiver* para que al recibir mensajes del tipo *"/conectado"* si ya se encontraba en la lista de conectados llame a la función *sigueConectado* que se acaba de comentar.

Con todo lo anterior se consigue el funcionamiento del indicador de conexión y al haber generado la base necesaria para enviar mensajes del servidor a los clientes abre posibilidades para nuevas mejoras.

6.5. Limitación del número de mensajes de los *scrollbar*

Se ha introducido un nuevo diccionario del tipo *Dictionary <string, float>* en el *script InstrumentController* el cual controlará el último valor enviado por cada uno de los *scrollbar*. Se ha introducido una condición en la función que envía los mensajes para que solo se realice cuando la diferencia con el último valor enviado sea por lo menos del mínimo establecido, reduciendo así los constantes envíos de mensajes sin cambios significativos.

6.6. Introducción de máquina de estado y sistema de colores

Se ha decidido implementar una forma de indicar los diferentes estados por los que pasa un intérprete, para los cuales se han tenido en cuenta que esté recién conectado, haya seleccionado su primer instrumento, modificado el anterior instrumento y que se haya establecido una configuración para este. Además se pretende estandarizar el sistema de colores que usen todos los botones, anteriormente usando enumeradores repetidos en diversos *scripts*.

Para ello se ha generado un *script* el cual forma una nueva estructura denominada *EstadoColor* en este *script* se han declarado tantos enteros como estados posibles, todas las variables son del tipo constante, ya que son usadas para controlar una máquina de estados, y se ha generado una función la cual traduce los diversos estados a colores, que serán usados para colorear todos los botones.

Estado	**Color**
Estado normal de los botones	
Botón seleccionado	
Nuevo interprete	
Primera elección de instrumento	
Modificación del instrumento	
Interprete configurado	

Tabla 4. Relación de colores y estado

Se han eliminado los antiguos enumeradores y *array* de colores empleados de los *scripts UIServidor* y *Organizador*, a los que se les ha realizado diversos cambio al igual que a *AvatarInterprete, GestorAvatar* y *OSCReceiver*.

Empezaremos por *AvatarInterprete* al cual se ha añadido una nueva variable entera llamada *estado,* que guarda la situación del intérprete dependiendo de los estados mencionados anteriormente, y otra del tipo *string* donde se guarda el nombre del botón *organizador* que le hace referencia.

La variable *estado* se modifica al ser instanciado el interprete, modificado el valor del instrumento y en una nueva función *guardar* que establece su estado al correspondiente.

En *OSCReceiver* se ha modificado la estructura interna para añadir dos nuevos parámetros y controlar así la asignación de nuevos instrumentos de forma similar a como se realiza cuando se crea un nuevo interprete, explicado en el apartado 0.

Los cambios realizados a *GestorAvatar* se limita a la llamada a *setInstrument* en el cual se guarda el antiguo instrumento antes de modificarlo y se llama a una nueva función de *UIServidor*.

Respecto a *Organizador* se ha eliminado el método *seleccionado y* la variable que guardaba el componente botón, manteniendo el objeto para cuando sea necesario. Se ha modificado el *set* de interprete para que cuando se establezca, si es distinto de nulo,

se guarde el nombre del botón que lo representa en su estructura interna y se establece el color del botón organizador según es estado actual del intérprete.

Los otros métodos modificados de este *script* son *posicionar, reiniciar* y *reiniciarColor*. Ahora *posicionar* llama a *reiniciarColor* antes de concluir, *reiniciar* ha sido modificado para que si contenía a un intérprete éste eliminase la información sobre este organizador que le hace referencia y establece el color del botón al normal y *reiniciarColor* ha sido cambiado para que si no referencia a ningún interprete se establezca el color normal y en caso contrario el correspondiente al del estado del intérprete.

Por último respecto al *Organizador* se ha implementado un nuevo método que llama al anteriormente mencionado *guardar* de *AvatarInterprete* y posteriormente llama al método propio *reiniciarColor*.

Para el funcionamiento correcto de todo lo mencionado ha habido que realizar algunos cambios en *UIServidor* sustituyendo las antiguas referencias al enumerador y lista de colores así como las llamadas al método *seleccionar* de *Organizador* por llamadas al nuevo sistema.

También se ha generado un método en *UIServidor* como se mencionó el cual es llamado desde *GestorAvatar* para actualizar la información de la interfaz. Este nuevo método *instrumentoSeleccionado(string idOrganizador)* comprueba el estado actual del intérprete, si su estado es que se ha modificado actualiza el color del botón al correspondiente, en caso contrario se actualiza al de instrumentos seleccionado por primera vez.

6.7. Reinicio de efectos

Se ha introducido un método en *UIServidor* que reinicia la asignación de efectos. Es usado para evitar perder la referencia a los efectos cuando un instrumento de interprete sea reemplazado por otro.

El método *liberaEfectoAudio(string idOrganizador, string id, int antiguo)* será llamado desde *instrumentoSeleccionado(string idOrganizador, string id, int antiguo)* -el cual tendrá dos argumentos de entrada adicionales para ello- dentro de la comprobación de que estado del intérprete sea "modificado".

Se ha pensado en la posibilidad de que se trate de un instrumento que está siendo configurado, si ese fuera el caso se finalizaría la configuración de este, una vez hecho esto se llamaría a la función creada en *AvatarInterprete reiniciaEfectos* del intérprete que se está viendo afecto, este método desasignará cualquier efecto que tuviera guardando los nombres de aquellos que sean efectos de audio en una lista que se devuelve como resultado, está lista es usada por *UIServidor* en la función mencionada para liberarlos ya que son de asignación única y se conserva una lista con los asignados.

6.8. Sistema para selección de distintos avatares

Se ha querido implementar un modo de poder seleccionar para cada intérprete un avatar entre un conjunto de ellos. Se ha realizado de modo que los avatares pueden tener distintos efectos visuales los cuales no serían seleccionables por otros avatares y también que esté la posibilidad de compartir ciertos efectos entre ellos.

Para dar al usuario la posibilidad de seleccionar entre los distintos avatares se ha generado un nuevo menú en la interfaz el cual creará un nuevo botón por cada posible avatar a seleccionar y mostrará una imagen representativa de estos, facilitando así la identificación de cada uno.

Ilustración 48. Menú avatares

Una vez se seleccione uno de ellos este se volverá de color verde y habilitará aquellos efectos disponibles para dicho avatar. En el caso de que se seleccione otro avatar, el anterior será cambiado por el nuevo y todos los efectos de video que fueron asignados serán reiniciados, teniendo que reasignar estos.

Para que esto funcione ha habido que realizar un gran número de cambios al prototipo que se desarrolló empezando por añadir una nueva función al *HelpScript* la cual se encarga de generar el nuevo menú sin necesidad de modificar la interfaz manualmente. Para ello se ha generado un método denominado *menuDinamicoH*, el cual al recibir el objeto en el que interactuar, una lista de nombres, un valor indicando si se añadirán imágenes y el directorio de éstas genera un botón por cada uno de estos nombres y añade las respectivas imágenes, las cuales tendrán como nombre los pasados como lista.

Se ha modificado la estructura de los intérpretes para poder introducir esta nueva característica y que fuera independiente a los efectos de audio. Ahora se genera un objeto en el cual se agregan aquellos elementos ajenos al propio avatar como son los *scripts* de audio y red y se instanciará por otro lado el avatar que se seleccione, que para controlar se han implementado tres nuevos métodos

- *setAvatar(string avatarName):* genera el avatar cuyo nombre se ha proporcionado, en caso de existir otro con anterioridad este se eliminaría, se agregan los *scripts* de control de efectos y se posiciona el avatar en su correspondiente lugar.

- *getAvatarName()*: si dispone de un avatar devuelve su nombre.

- *reiniciaVideo ():* reinicia todos los efectos de video previamente asignados.

Con este nuevo sistema ya no es necesario que se instancien avatares desde el *GestorAvatar* por lo que se ha eliminado y dejado está funcionalidad en manos de *UIServidor* haciendo uso de los ya comentados métodos de *AvatarInterprete*.

Otro de los aspectos que se querían lograr era poder independizar efectos entre distintos avatares por lo que se realizará un nuevo *script* de interfaz en el que se

expondrán aquellos efectos de los cuales puede hacer uso cada uno de ellos y que serán implementados desde *ControladorVideo*.

Ahora que ya se dispone de la base suficiente para seleccionar los avatares y que sean independientes entres sí se ha otorgado el control a *UIServidor*. Se ha declarado las diferentes variables que serán necesarias que son

- Listas de *strings* por cada avatar y otra para efectos globales, para controlar los efectos que tiene disponible cada uno.

- Lista con los nombres de los distintos avatares.

- Objeto que referencia al panel donde se irán insertando los botones, los cuales se guardarán en otra lista.

- D*ictionary<string, Button>* para dar diferentes comportamientos a estos respecto a los de audio.

Con estas variables se ha modificado la función *Start()* para adaptarla. Se han inicializado las respectivas variables, llamando a la funciona de *menuDinamicoH* y guardando la lista de botones, a los que se les ha asignado como respuesta a los *click* la función *selectedAvatar* que se explicará más adelante.

En esta misma funcion *Start* se han realizado cambios en la inicialización de las variables de video, añadiendo cada uno de los efectos generales y de avatares al diccionario comprobando que no se repita ninguno de ellos, ya que pueden compartir entre avatares, una vez realizado se rellena el menú de efectos de video con el resultado obtenido y asigna el mismo método que anteriormente tenia pero con algunas modificaciones.

Otro de los cambios sufridos en *UIServidor* ha sido en el asignado al botón de seleccionar un intérprete de la ventana organizadora, al tener que controlar la situación del menú de avatares al ser seleccionado.

Si se selecciona un intérprete con instrumento seleccionado se debe de comprobar si dispone de un avatar y si fuera el caso marcarlo en color verde y deshabilitarlo, una vez se posee un avatar no puede quedarse vacio.

En el método *finalizaConfiguracion* se ha tenido en cuenta que se debe de establecer el menú de avatares a un estado independiente de los intérpretes por lo que al finalizar una configuración se reinician los colores y deshabilitan.

selecInsturment ahora antes de habilitar los efectos de video comprueba de que se haya seleccionado un avatar y de ser así solo habilita los efectos generales y aquellos que sean del propio avatar, los cuales tienen la misma funcionalidad, pudiendo ser seleccionados o eliminada dicha asignación volviendo a *clickear* el mismo botón.

En el caso de seleccionar uno de los avatares en su respectivo menú se activará el método *selectedAvatar*. Este nuevo método implementado se encarga de hacer las debidas llamadas a las funciones del intérprete para comprobar si existía previamente una selección y de ser así cambiar su color, habilitar dicho avatar como seleccionable en

su menú y deshabilitar aquellos efectos que le pertenecieran. Si había algún efecto seleccionado se reinicia su color al normal y habilitan todos aquellos efectos que sean aplicables al nuevo avatar.

Por último se reinician los efectos de vídeo y establece el nuevo avatar en el propio interprete, indicando cual ha sido seleccionado deshabilitando y coloreando de verde el botón correspondiente.

6.9. Representación gráfica del resultado

Se ha insertado una forma de representar el resultado del audio final obtenido siguiendo la guía y código proporcionado por (DimasTheDriver, 2015). Ha sido ligeramente modificado para en lugar de usar el espectro de la fuente de audio se uso el del *AudioListener* obteniendo una representación gráfica de los resultados finales.

7. Conocimientos aplicados

Para realizar un proyecto como este hace falta conocimientos de diferentes ramas de la informática. A la hora de programar hay que tener conocimientos de estructuras de datos, diseño de interfaces, usabilidad, redes,... Además de ciertos conocimientos en otros aspectos como son diseño 3D, edición de animaciones y conocimientos de audio entre otros para poder comprender y aplicar las diferentes funciones de la aplicación. Durante el proyecto hay aplicadas muchas de las técnicas aprendidas en diversas asignaturas que se han ido aplicando.

Respecto a la programación, que se comienza a estudiar en primero con **introducción a la informática y la programación** haciendo uso de Java, hay que agradecer también a otras asignaturas como **concurrencia y sistemas distribuidos**, la cual enseña a hacer uso de *multithreading* y que los problemas planteados exigían de un alto nivel de exigencia lo cual personalmente siempre me gustó, y las prácticas realizadas de **computabilidad y complejidad** y **sistemas de almacenamiento y recuperación de información** que fueron de gran ayuda a la hora de mejorar las habilidades y facilitar posteriormente el enfrentarse a nuevo retos de la programación haciendo usos de diccionarios y algoritmos de cierto grado de complejidad. Algo de lo que no hay que olvidarse es también de la lógica aprendida en **matemática discreta** ya que sin lo aprendido en ella sería complicado realizar cualquier código.

Para la selección de la mejores estructuras de datos siempre recuerdo lo aprendido en **estructuras de datos y algoritmos**, que explico cómo componer diferentes estructuras de datos complejas a partir de las, y en **sistemas de almacenamiento y recuperación de información** las cuales fueron las que hicieron uso de pilas, colas, listas y diccionarios.

En cuanto a redes se refiere para comprender el funcionamiento de la tecnología *OSC* empleada, así como la posterior implementación realizada, se ha usado lo aprendido en **tecnologías de sistemas de información en la red** que explica el funcionamiento de estas y las diferentes estructuras a emplear dependiendo de lo que necesite la aplicación en desarrollo, en este caso se trata de una aplicación servidora a la cual se conectan un número indeterminado de clientes.

En el diseño de interfaces y estudio de usabilidad ha tenido un peso fundamental la asignatura **interfaces persona computador** que es la primera en salir del terminal para mostrar funcionalidades externas, son los primeros pasos que se hacen en dar funcionalidad a dichas interfaces haciendo uso del *QT Jambi*. Ha sido empleado para desarrollar las interfaces de móvil y PC.

Otra asignatura que también han tenido un gran peso es **introducción a los sistemas gráficos interactivos** la cual explica cómo se dibuja por pantalla y la forma de programación a seguir para que ello se pueda realizar, lo cual ha ayudado a entender la forma de trabajar con *Unity*, el cual usa los métodos de *Update* y *FixedUpdate* para realizar los cambios gráficos, de forma similar a las prácticas de dicha asignatura, en las cuales se iniciaba un bucle que se encargaba de dibujar cada *frame* y mantener actualizada la información.

Además de todas estas, las asignaturas de **arquitecturas y entorno de desarrollo para videoconsolas**, **integración de medios digitales** e **introducción a la programación de videojuegos,** han ayudado a conseguir confianza para enfrentarse a nuevos retos ofreciendo la posibilidad de seleccionar, gestionar y realizar proyectos propios para dichas asignaturas, lo que de cara a realizar el proyecto final de grado a supuesto una gran ayuda. En ellas también se han explicado las últimas tecnologías más empleadas y la forma de programar cuando se trata de grandes proyectos. Por ejemplo de **introducción a la programación de videojuegos,** se ha usado los conocimientos de máquinas de estado para realizar una programación más simple y comprensible visto desde el exterior lo que mejora el entendimiento, la modificación de alguno de los estados en particular y la inclusión de nuevos a las creadas con anterioridad.

8. Pruebas de ejecución

Durante el desarrollo de la aplicación se han ido realizando diversas ejecuciones comprobando el correcto funcionamiento de las distintas características que se han ido desarrollando.

Una vez concluidas ambas aplicaciones se ha realizado una prueba de ejecución con lo ayuda de dos compañeros en busca de posibles errores en alguna de las aplicaciones o saturación del ancho de banda. Se han asignado distintos avatares a cada uno de ellos y asignado el control de diferentes efectos de audio, video y generación de sonidos.

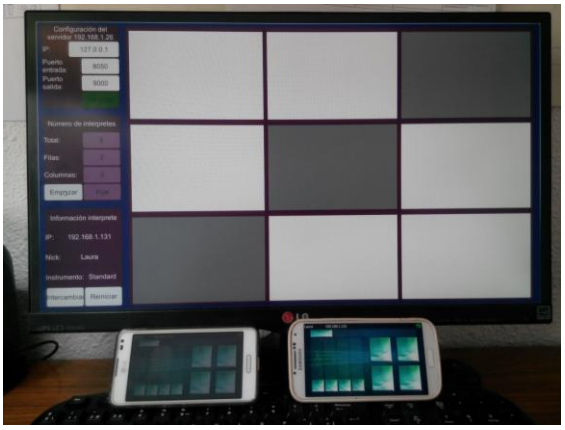


Ilustración 49. Configuración de los terminales

También se ha incluido una canción de fondo para poder apreciar mejor los diferentes efectos de audio que están disponibles.

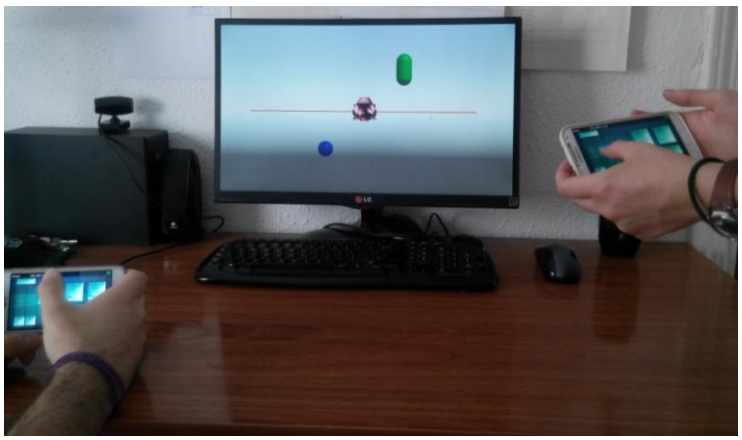


Ilustración 50. Ejecución con colaboración

Una vez configurados se ha realizado una interpretación libre observando que no se ha producido ningún tipo de error ni en las aplicaciones móviles ni en la de *PC*, pudiendo determinar que con tres personas interpretando la conexión de red permanecía estable y no se apreciaban respuestas tardías producto de una sobrecarga de la conexión.

No se ha podido realizar pruebas con un mayor número de móviles para localizar los límites en los cuales podría llegar a fallar, aunque es un dato que sería importante conocer y se deja como una tarea futura a realizar.

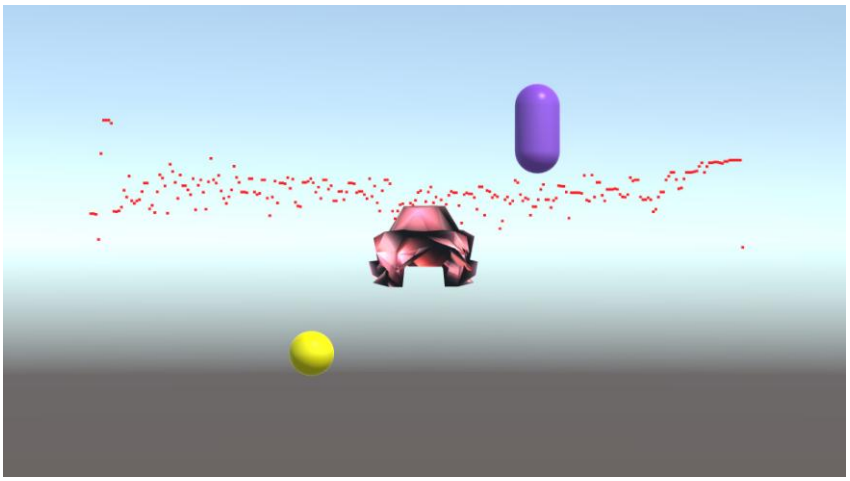


Ilustración 51. Instantánea de la ejecución

9. Mejoras futuras

Lo realizado hasta el momento es una versión inicial desde la cual seguir desarrollando, tanto modificando lo ya presente como añadiendo nuevas características.

Al ser un programa dedicado a la configuración de una orquesta los parámetros que pueden modificarse son muy amplios, yendo desde la gestión de los intérpretes a la asignación de distintos efectos multimedia, lo que implica un número casi ilimitado de posibilidades.

Las mejoras pueden ser modificaciones de la interfaz, inserción de nuevos sonidos, efectos de video y audio, implementación de nuevas funcionalidades... Algunas de las ideas para aumentar la calidad de aplicación serían las siguientes

- Introducción de menú para la personalización de generadores de audio
- Mecanismo para permitir la configuración de los máximos y mínimos de los distintos efectos de audio, video y generadores de audio
- Forma de guardado de distintas configuraciones para conservar aquellas ya establecidas y poder cargarlas posteriormente
- Inclusión de animaciones de objetos mediante *Animators*
- Mejoras visuales en las interfaces
- Menú de selección con previsualizaciones de los distintos efectos de video y avatares
- Elección de distintos temas para las interfaces
- Efectos visuales de entorno
- Implementación de distintos efectos de cámara
- Mejoras en la visualización de los resultados finales
- Test de usabilidad de las interfaces
- Test de latencia y determinación de los límites

10. Conclusión

Teniendo presente los objetivos iniciales a cumplir, al llegar a este punto y ver cómo no solo han sido cumplidos sino que además se han conseguido ampliar añadiendo diferentes características sobre el prototipo inicial se siente una gran satisfacción personal.

Todo ello se ha realizado con un código modular y escalable que facilita el poder seguir trabajando en el futuro. Durante el transcurso del proyecto han surgido una gran cantidad de dudas como a la hora de coordinar ambas aplicaciones, teniendo que controlar desde la parte del servidor a todos los clientes que se conectasen haciendo uso del protocolo OSC para las comunicaciones y diccionarios y estructuras de datos para la persistencia se ha logrado gestionar de forma adecuada y se ha conseguido formar una base de la cual partir para desarrollar los aspectos visuales y sonoros, además de poder incluir nuevas funcionalidades.

La realización de este proyecto ha servido para darse cuenta de todo lo aprendido durante estos años sin lo cual no se podrían haber realizado las aplicaciones necesarias para lograr los objetivos establecidos. Todo lo realizado en este proyecto ha sido pensado para poder seguir desarrollando en un futuro diferentes aspectos de usabilidad, interfaces, funcionalidades, efectos... para lo que se han aplicado todos los conocimientos posibles adquiridos durante la carrera llegando a formar una solida base desde la cual partir en nuevos proyectos.

11. Bibliografía y recursos

DimasTheDriver. (21 de Junio de 2015). Obtenido de http://www.41post.com/4776/programming/unity-making-a-simple-audio-visualization: http://www.41post.com/4776/programming/unity-making-a-simple-audio-visualization

Fundación Wikimedia, Inc. (s.f.). *Wikipedia*. Recuperado el 2015 de Marzo de 17, de OpenSound Control: http://es.wikipedia.org/wiki/OpenSound_Control

Heavers, M. (s.f.). *GitHub*. Recuperado el 20 de Marzo de 2015, de unity osc receiver: https://github.com/heaversm/unity-osc-receiver

Hen, A. L. (22 de Abril de 2015). *Develop-online*. Obtenido de http://www.develop-online.net/tools-and-tech/procedural-audio-with-unity/0117433

Herraez, J. B. (2013/2014). *Videojuego Beast's Retreat en Unity con C# integrando RT-DESK*. Valencia, Valencia, España: Escola Tècnica Superior d'Enginyeria Informàtica.

Muzykov, K. (16 de Marzo de 2015). *Raywenderlich*. Obtenido de http://www.raywenderlich.com/78675/unity-new-gui-part-1

Rose, M. (s.f.). *Gamasutra*. Recuperado el 19 de Marzo de 2015, de FMOD Studio audio tools now completely free for indies: http://www.gamasutra.com/view/news/212696/FMOD_Studio_audio_tools_now_completely_free_for_indies.php

SoundCool. (s.f.). *SoundCool*. Recuperado el 8 de Marzo de 2015, de http://soundcool.net/es/

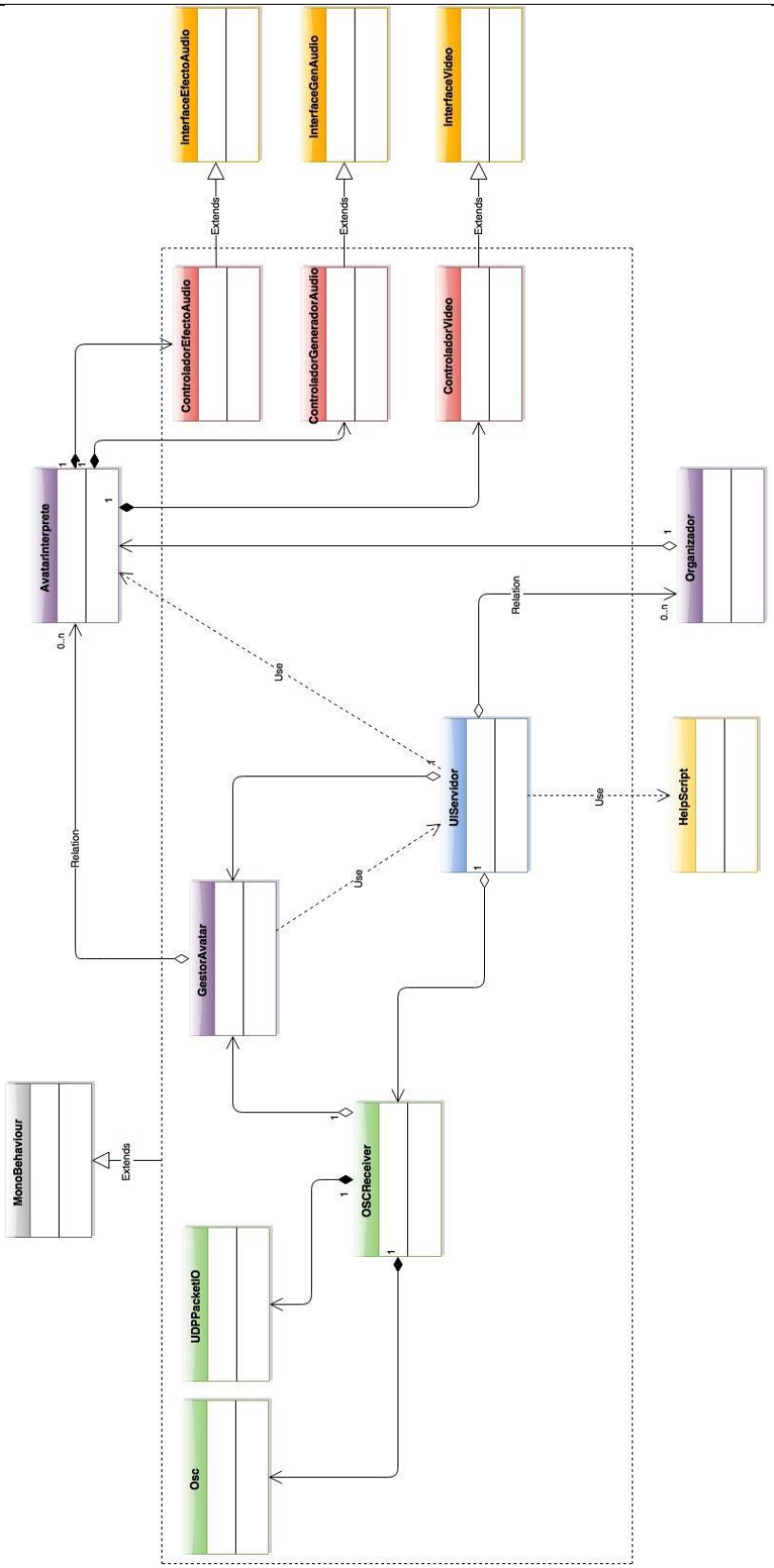
Unity Technologies. (s.f.). *Unity Documentation*. Recuperado el 16 de Marzo de 2015, de Execution Order of Event Functions: http://docs.unity3d.com/Manual/ExecutionOrder.html

Fondos e imágenes empleadas

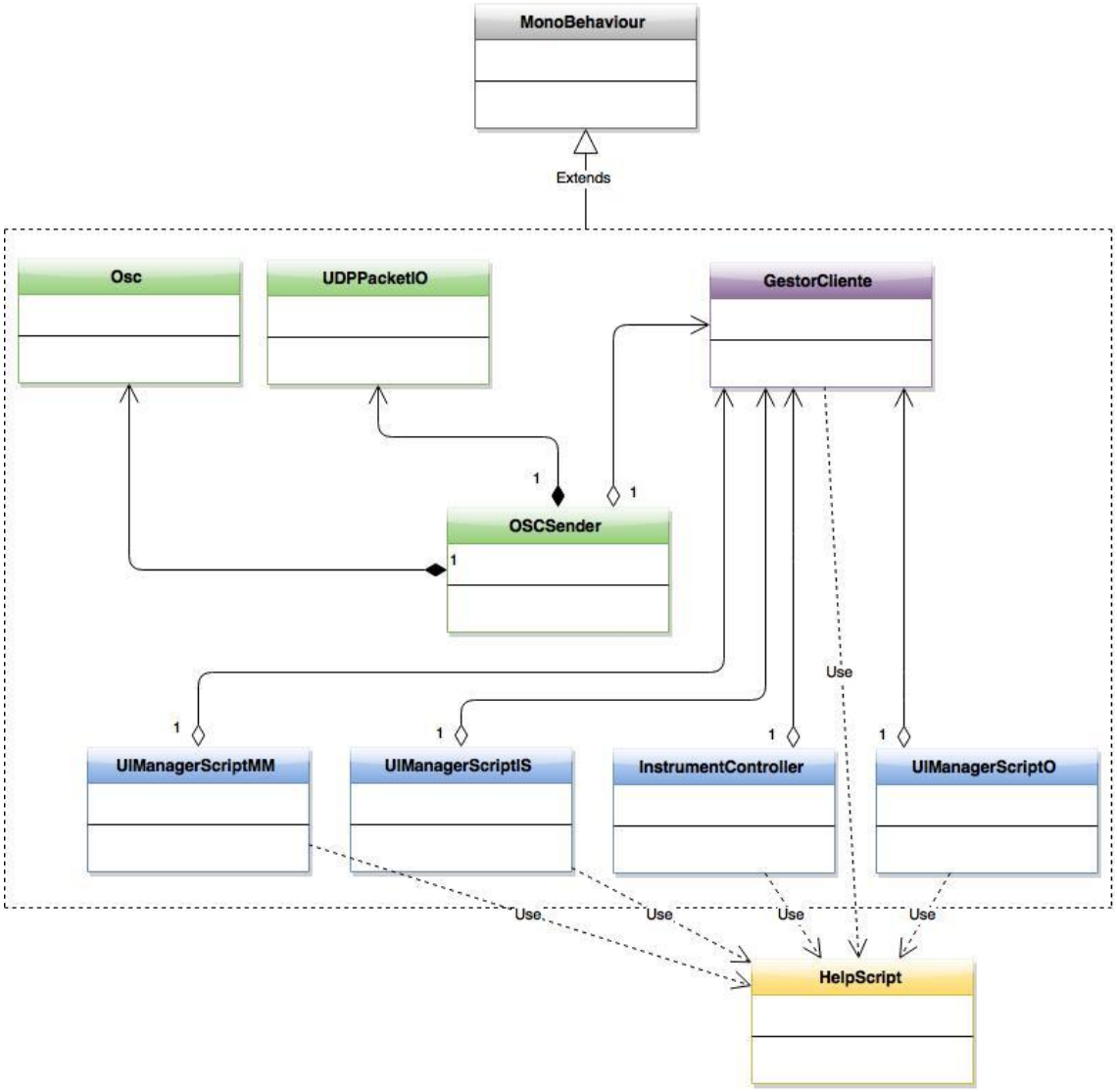
http://findicons.com/

http://www.icons-land.com/

Anexo I. Diagrama del servidor completo



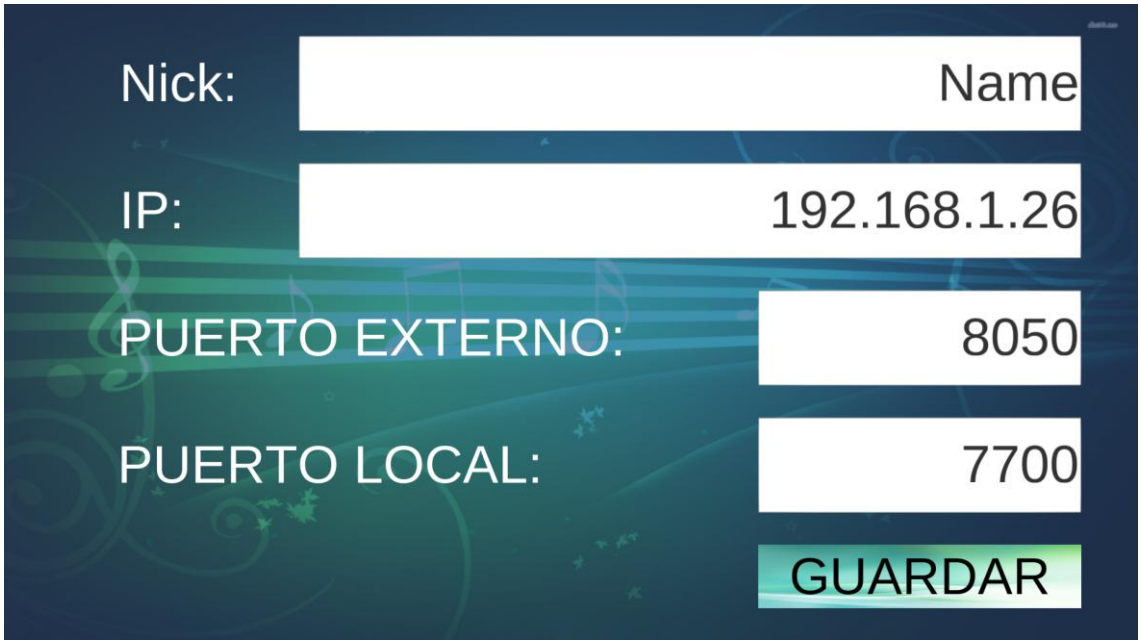
Anexo II. Diagrama de clases cliente



Anexo III. Nueva interfaz cliente

Menú principal

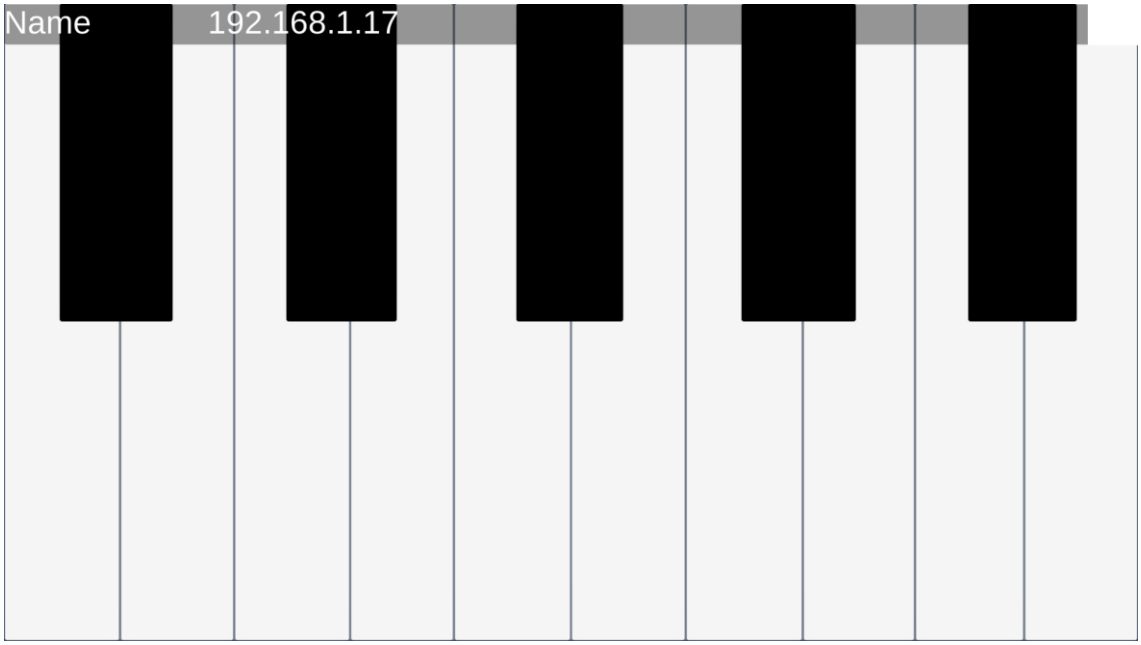
Opciones



Selección de instrumentos

Panel instrumental *standard*

Panel instrumental teclado



Panel instrumental percusión

Anexo IV. Manual del usuario

Para configurar correctamente la aplicación lo primero es, una vez abierta la aplicación servidora, configurar el puerto al cual se conectarán los diversos usuarios clientes, la dirección *IP* de salida puede ser cualquiera y el puerto de salida debería ser o inferior al de entrada o mayor que el puerto de entrada más el número de interpretes que se vayan a conectar. Si se establece el puerto 8000 y se van a conectar 13 interpretes el puerto de salida deberá ser menor que 8000 o mayor que 8014. Una vez realizado esto ya se puede pulsar el botón de conectar y seguir con el siguiente paso.

Ahora se deberá de establecer el número total de interpretes que se van a poder conectar y en cuantas filas y columnas se podrán organizar, en caso de introducir números incorrectos se reiniciarán los datos de entrada. Cuando se hayan introducido los datos deseados se pulsará la tecla de 'Fijar' y se crearán tantos botones como filas por columnas se indicaran.

Realizados estos dos pasos llega el turno de que se conecten los diversos clientes para lo cual deberán de acceder al panel de opciones y configurar los diferentes parámetros. En 'Nick' se introducirá un nombre identificativo para el usuario del *PC*, en *IP* deberán introducir la indicada en la aplicación servidora en el primer panel 'Configuración del servidor xxx.xxx.xxx.xxx', puerto externo el configurado en 'Puerto entrada' y el puerto local cualquier otro que no esté siendo usado, tras rellenar todos los datos se pulsa en 'GUARDAR', 'Play' y se selecciona un instrumento.

Tras esto la labor del intérprete se basará en usar los diferentes controles para realizar la interpretación una vez el usuario del servidor le dé comienzo.

Conforme se van conectando los distintos interpretes los botones irán cobrando distintos colores que tendrán los siguientes significados:

Estado	**Color**
Estado normal de los botones	
Botón seleccionado	
Nuevo interprete	
Primera elección de instrumento	
Modificación del instrumento	
Interprete configurado	

Si se encuentran en alguno de los tres estados inferiores al ser pulsados se pasará al panel de configuración del intérprete donde habrá que asignarle un avatar e ir asignado a cada uno de los diferentes botones del panel instrumento los efectos que vayan a ser usados por dicho interprete.

Si durante la configuración algún interprete cambia de instrumento se reiniciarán todos los efectos que tenia asignados y si se realiza un cambio de avatar se reiniciarán únicamente aquellos que sean efectos visuales.

Otra de las opciones disponibles es la de intercambiar las posiciones de los distintos avatares entre las filas y columnas que representan posiciones de avatares una vez comienza el espectáculo. Para realizar un intercambio se debe pulsar el botón de intercambiar y a continuación dos de los botones de intérpretes, lo que realizará el intercambio. Otro de los botones presentes es el de reiniciar el cual eliminará todos los intercambios realizados.

Tras finalizar la configuración de cada uno de los intérpretes todos ellos aparecerán en color gris y se podrá pulsar el botón de 'Empezar' para ocultar el menú de opciones y mostrar los avatares.

Otras teclas de interés son la 'M' para volver a mostrar el menú de opciones, 'Esc' para cerrar la aplicación, las flechas de dirección para mover la cámara en los ejes 'x' e 'y' y las teclas '+' y '-' para acercarla o alejarla, eje 'z'.