ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

“Desarrollo de aplicaciones con Microsoft Kinect”

Iñaki Iralde Lorente

Alfredo Pina Calafi

Pamplona, 23/04/2012

Agradezco a mi tutor Alfredo Pina, que me diera la oportunidad de trabajar en este proyecto, que me ha dado la oportunidad de trabajar con una tecnología innovadora en un campo en el que no había trabajado nunca. Gracias a sus consejos y entusiasmo el proyecto ha tenido unos resultados que de otro modo no habrían sido posibles.

Agradezco a Isabel Sánchez Gil, sus consejos como experta en educación que fueron de gran utilidad a la hora de perfilar el prototipo de aplicación educativa, así como por darme la oportunidad de poder realizar una aplicación que será expuesta al público en su exposición.

Agradezco a Benoît Bossavit su implicación en el proyecto, así como los consejos que me han permitido hacer un mejor trabajo.

Quiero agradecer a Asier Marzo sus consejos, sus ideas han influido mucho en los resultados finales del proyecto.

Agradecer a nuestros "beta-testers" Carlos, Maialen y Paula su colaboración en las pruebas de usuario realizadas.

Finalmente quiero agradecer a mis padres y hermana su apoyo y en sí haberme aguantado todo este tiempo.

Tabla de Contenidos

TABLA DE ILUSTRACIONES

Capítulo 1 Introducción

La reciente aparición de la herramienta Microsoft Kinect ha sido un rotundo éxito en el mundo de la informática, numerosos equipos de todo el mundo están desarrollando diferentes aplicaciones para campos cada vez más variados, separándose del campo de los videojuegos que era su objetivo. Microsoft (y otras compañías) consciente de este éxito, ha publicado un SDK oficial dirigido a comunidades académicas con el objetivo de facilitar la investigación de las diferentes posibilidades que ofrece la herramienta.

Dentro de este proyecto se tienen los siguientes objetivos:

- Aproximación a la herramienta Microsoft Kinect, se desea conocer la herramienta Kinect, como es, que posibilidades ofrece, con que tecnologías se puede combinar para tener mejores resultados. En definitiva tener una primera idea de las aplicaciones que se podrán realizar y como funcionarán estas.
- Ver posibles campos de aplicación., la novedad de la forma de interacción que ofrece Kinect, hace que sea una herramienta que tenga aplicación en infinidad de campos. Se desea, a partir del conocimiento adquirido en el punto anterior, buscar algún campo donde se pueda aplicar Kinect de manera innovadora.
- Desarrollo de aplicaciones, para algunos de los campos seleccionados en el punto anterior, se deseará desarrollar alguna aplicación que muestre cómo es posible aplicar Kinect en ese campo de trabajo.

A lo largo de la memoria se tratará de dar respuesta a los puntos anteriores, la estructura de la memoria será la siguiente:

- Capitulo 2 Microsoft Kinect, se hace una descripción del hardware del dispositivo Microsoft Kinect.
- Capítulo 3 Kinect SDK Beta2, en este capítulo se explicarán muchos de los aspectos más relevantes del SDK que se utilizará para desarrollar las aplicaciones.
- Capitulo 4 Primeras mini-aplicaciones, con el objetivo que conocer y coger destreza en el desarrollo de aplicaciones con Microsoft Kinect y su SDK beta2 se desarrollaron cuatro mini-aplicaciones que implementaban algunas de las características más importantes de Kinect (seguimiento de esqueleto, detección de movimiento, uso de cámara de profundidad y RGB…), en este capítulo se hace una descripción de estas aplicaciones y de su funcionamiento.
- Capítulo 5 Kinect aplicado a la educación infantil, se va a describir la aplicación desarrollada en el campo de la educación infantil así como las diferentes formas de interacción estudiadas (junto con sus resultados).
- Capítulo 6 Kinect aplicado en el mundo del arte, en este capítulo se tratara la aplicación desarrollada para la exposición de Isabel Sánchez Gil “cuando el umbral diferencial no es visible”.
- Conclusiones y líneas futuras.
- Apéndice 1, paper presentado al congreso IDC 2012 que contiene una descripción del sistema las pruebas y sus resultados de la aplicación desarrollada en el capítulo 5.
- Apéndice 2, descripción de Isabel Sánchez Gil y descripción de su obra y exposición.

Capítulo 2 Microsoft Kinect

Figura 2.1 Imagen Kinect

Kinect es un dispositivo de control por movimiento creado originalmente para jugar a los videojuegos de Xbox360 sin necesidad de ningún mando o controlador, Kinect da la oportunidad de interactuar con el cuerpo en el videojuego mediante una interfaz natural de usuario que reconoce gestos, comandos de voz e imágenes. Debido a sus características Kinect compite directamente con otros sistemas como Wiimote o PlayStation Move de Nintendo y Sony respectivamente.

Kinect fue anunciado por primera vez en junio de 2009 bajo el nombre de "Project Natal". El 13 de junio de 2010 se reveló que el nombre final sería Kinect y el 4 de noviembre de 2010 salió a la venta en Estados Unidos y México. Para su promoción en Estados Unidos Microsoft destinó un presupuesto de 500 millones de dólares (una suma mayor que la inversión en el lanzamiento de la Xbox).

Al poco tiempo de su publicación la empresa Adafruit ofreció una recompensa de 2000$ a la primera persona que consiguiera hackear Kinect, el ganador fue Héctor Martin (el 10 de noviembre de 2010), un mes más tarde la empresa PrimeSense lanzó el primer SDK no oficial para Kinect.

Durante la primera mitad del 2011 numerosos desarrolladores, instituciones (como el MIT), etc. empezaron a investigar y programar nuevas aplicaciones que pudieran aprovechar las características del Kinect para darle un uso que vaya más allá de los videojuegos.

Debido a este éxito y con el objetivo de canalizarlo Microsoft publicó la primera beta de su SDK oficial compatible con Windows 7 el 16 de junio de 2011. La licencia de esta beta es no-comercial aunque en 2012 se espera una versión comercial.

El aumento de aplicaciones para PC ha hecho que Microsoft anuncie que en 2012 publicará un Kinect orientado para su uso en PC.

2.1 Partes fundamentales del Hardware

Se pueden distinguir cuatro partes fundamentales dentro de los componentes hardware:

Figura 2.2 Partes Hardware

Kinect se trata de una barra de plástico negro de 30 cm de ancho conectada a un cable que se bifurca en dos, un cable USB y otro un cable eléctrico.

- **Cámara RGB,** cámara de video con una resolución de 640x480 a 30 fps.
- **Sensores 3D de profundidad,** combinación de un proyector de profundidad (retícula izquierda) con un sensor de profundidad (retícula derecha), se calcula la distancia en función del tiempo que tarda en reflejar la luz.
- **Inclinación monitorizada,** permite ajustar la cámara hacia arriba o hacia abajo hasta 27º.
- **Micrófono Multi-array,** conjunto de cuatro micrófonos que se monta como un solo micrófono.

Y aunque no visibles a simple vista, Kinect también posee:

- **Memoria RAM de 512 Mb**
- **Acelerómetro,** para estabilizar la imagen cuando se mueve.
- **Ventilador,** no está encendido continuamente para no interferir con los micrófonos.

2.2 Funcionamiento

Se puede dividir el funcionamiento de Kinect en tres partes:

- Reconocimiento de imágenes.
- Reconocimiento de voz.
- El motor

2.2.1 Reconocimiento de imágenes

La configuración óptica permite el reconocimiento de imágenes en tiempo real. Kinect no usa tecnología compleja, de hecho la tecnología usada está disponible desde hace 15 años, pero Microsoft ha conseguido efectos y funciones que antes estaban disponibles solamente con un gran costo.

Podemos dividir dos partes principales, el proyector y la cámara de infrarrojos VGA. El rebote de los haces de laser por todo el campo de juego es lo que permite que la cámara capte al profundidad de los diferentes objetos.

Con estos datos Kinect ejecuta una serie de filtros con la intención de calcular que es una persona y que no lo es. El sistema utilizará directrices como “una persona tiene una cabeza, dos piernas y dos brazos” para diferenciarla del sofá o de algún otro elemento que pueda haber en el campo de juego. También es capaz de distinguir si se usa ropa holgada o se tiene el pelo largo.

A partir de esta información, se ordena y convierte la identificación de las partes del cuerpo en un esqueleto. Kinect tiene precargadas más de 200 posiciones comunes del ser humano por lo que en caso de que alguna acción tape alguna parte del esqueleto a la cámara, Kinect llenará el vacio automáticamente, se generan varios esqueletos pero se elige uno basándose en la experiencia.

El sistema hace todo esto continuamente a una velocidad de 30fps y hay que estar a una distancia de unos dos metros para poder ser reconocido.

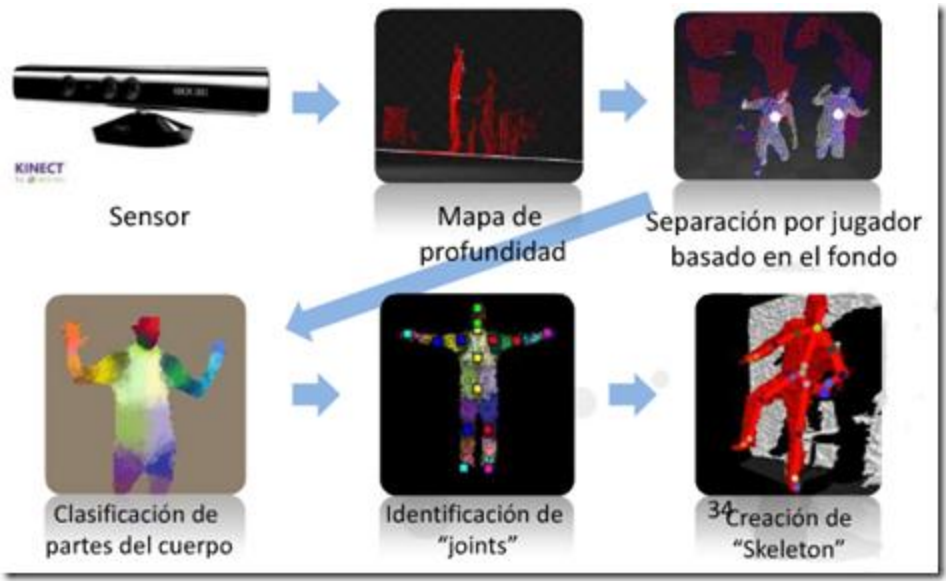


Figura 2.3 Reconocimiento esqueleto

2.2.2 Reconocimiento de Voz

El mayor problema del reconocimiento de voz era que tenía que ser sensible a voces de hasta cinco metros de distancia además de ignorar los ruidos ambientales y cualquier otro sonido. Para solucionarlo Microsoft puso en doscientas cincuenta viviendas dieciséis micrófonos para tomar una serie de grabaciones con el objetivo de determinar cuál es el mejor posicionamiento del micrófono.

Como resultado se tiene una colocación específica que hace que el Kinect sea tan ancho como es con un micrófono boca abajo, uno en la izquierda y tres en la derecha. Esta forma es la mejor para recoger las voces desde la distancia, el ruido asociado es cancelado por la unidad de procesamiento y se utiliza un sistema software que usa la cámara para calcular de donde viene el sonido y así crear un burbuja de sonido alrededor del usuario, de esta manera se separa el sonido de la voz y se hace caso omiso a las otras personas que se encuentren alrededor de los jugadores.

Al igual que el reconocimiento de imágenes, el reconocimiento de sonido está funcionando continuamente.

2.2.3 Motor

Tras investigaciones para ver las diferentes configuraciones de espacios de vida en toda América, Europa y Asia, Microsoft llego a la conclusión de que era necesario dotar la cámara del Kinect la posibilidad de moverse hacia arriba o hacia abajo con el objetivo de calibrar cada espacio concreto.

El motor es capaz de mover la unidad hacia arriba o hacia abajo más o menos unos 30º, por lo que la altura óptima está recomendada en uno o dos metros.

El motor también opera la cámara, el es quién activa la función de zoom, que permite ampliar el espacio de juego

Capítulo 3 Kinect SDK Beta 2

Tras la salida al mercado de Kinect no pasó mucho tiempo hasta que empezaron a hackear Kinect y apareciesen distribuciones libres como por ejemplo PrimeSense OpenNi. Este éxito provocó que Microsoft decidiese publicar la SDK oficial de Kinect.

El SDK está orientado a la investigación académica principalmente aunque también a programadores particulares con el objetivo que experimenten con la creación de interfaces naturales de usuario.

El SDK permitirá:

- Skeletal Tracking de uno o dos personas que estén en el ángulo de visión de Kinect.
- Cámara de profundidad que será capaz de calcular la distancia de los objetos al sensor de Kinect.
- Procesamiento de audio para sus cuatro micrófonos.

El SDK incluye:

- Drivers para usar Kinect en una computadora con Windows 7.
- APIs, interfaces de los dispositivos con documentación para desarrolladores.
- Ejemplos de código fuente.

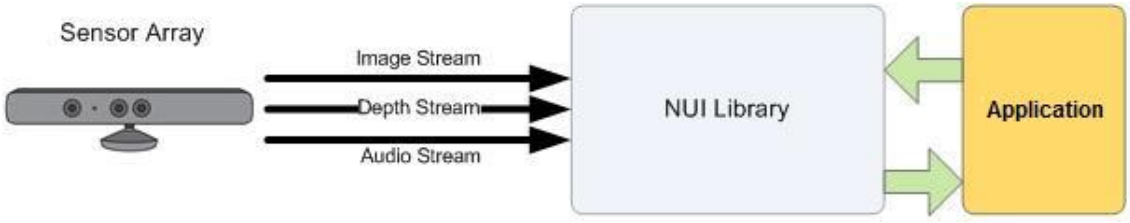


Figura 3.1 Interacción Hardware Software con la aplicación[1]

Microsoft no recomienda usar el SDK oficial de Kinect para el desarrollo de aplicaciones para la consola Xbox 360, debido a las diferencias arquitectónicas, de mantenimiento y de funcionamiento son distintas en Windows que en Xbox360. Para programación sobre Xbox se recomienda usar el Xbox depelopment kits (XDK).

[1] Programming Guide Microsoft Kinect SDK Beta 1 Draft Version 1.1

3.1 Requerimientos del sistema

Para ejecutar las aplicaciones desarrolladas con el SDK beta de Kinect será necesario utilizar un entorno nativo de Windows, esto quiere decir que no es posible ejecutar las aplicaciones en una máquina virtual ya que tanto los drivers de Kinect como el SDK deben estar instalados en la computadora donde está corriendo la aplicación.

Los lenguajes utilizados por el SDK son C# o C++.

- Sistema operativo: Windows 7 (32 o 64 bits).

- Requerimientos hardware:
  - Ordenador con un procesador dual-core 2.66 GHz (mínimo).
  - Tarjeta gráfica compatible con Windows 7 y que soporte Microsoft DirectX 9.0c.
  - 2Gb de memoria RAM.
  - Dispositivo Kinect.

- Requerimientos software:
  - *Microsoft Visual Studio 2010* (express o cualquier otra edición).
  - *Microsoft .NET Framework 4* (instalado con Visual Studio).

  - Para los ejemplos de esqueletos con C++:
    - *DirectX Software Development Kit* (versión junio 2010 o posterior).
    - *DirectX End-User Web installer*

  - Para el ejemplos de voz:
    - *Microsoft Speech Platform – Server Runtime*, versión 10.2 (para x86).
    - *Microsoft Speech Platform- Software Development Kit*, versión 10.2 (para x86).
    - *Kinect for Windows Runtime Language Pack*, versión 0.9.

3.1.1 Visual Studio 2010

Es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows, soporta lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones Web, así como servicios web en cualquier entorno que soporte plataforma .NET. De esta forma se pueden crear aplicaciones que se comuniquen entre dispositivos móviles, estaciones de trabajo y sitios web.

Desde la versión 2005 Microsoft viene ofreciendo gratuitamente las *Express Editions*, son ediciones básicas separadas por lenguajes de programación.

Microsoft también ha puesto gratuitamente a disposición del mundo una versión reducida de MS SQL Server (SQL Server Express Edition) cuyas principales limitaciones son que no soporta BBDD de tamaño superior a 4GB de tamaño, únicamente se ejecuta en un procesador y emplea un 1GB de RAM como máximo, tampoco cuenta con el Agente SQL Server.

La versión más reciente de este IDE es la versión 2010, que vio la luz el 12 de abril de 2010, su mayor logro ha sido incluir las herramientas para el desarrollo de aplicaciones para Windows 7.

Entre sus características se encuentran la capacidad para utilizar varios monitores así como la posibilidad para desacoplar ventanas de su sitio original y acoplarlas en otros sitios de la interfaz del trabajo. También ofrece la posibilidad de crear aplicaciones para otras plataformas de Microsoft como Azure, Windows Phone 7 o SharePoint, también es posible desarrollar aplicaciones que acepten pantalla táctil.

Las diferentes ediciones de Visual Studio 2010 son:

- **Visual Studio 2010 Ultimate**
- **Visual Studio 2010 Premium**
- **Visual Studio 2010 Professional**
- **Visual Studio Team Foundation Server 2010**
- **Visual Studio Test Professional 2010**
- **Visual Studio Team Explorer Everywhere 2010**

La licencia de Visual Studio es licencia propietaria.

3.1.2 C#

C# es un lenguaje de programación multiplataforma orientado a objetos desarrollado y estandarizado por Microsoft como parte de la plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO. C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

La sintaxis deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar a la de Java aunque con mejoras derivadas de otros lenguajes.

Aunque C# forma parte de la plataforma .NET, esta es una API, mientras que C# es un lenguaje de programación creado para realizar programas para esta plataforma. Existe ya un compilador creado para generar programas para distintas plataformas como Windows, Linux y GNU/Linux.

En la actualidad existen los siguientes compiladores para el lenguaje C#:

- **Microsoft .NET Framework (SDK),** incluye un compilador C#.
- **Microsoft Visual Studio.**
- **SharpDevelop,** IDE libre bajo licencia GNU LGPL, con interfaz similar a Visual Studio.
- **Mono,** es una implementación con licencia GNU GPL de todo el entorno .NET desarrollado por Novell.
- **Delphi 2006,** de Borland Software Corporation.
- **DotGNU Portable .NET,** de la Free Software Foundation.

3.1.3 Microsoft .NET

Se trata de un framework de Microsoft que hace énfasis en la transparencia de redes con independencia de la plataforma hardware y que permita un rápido desarrollo de aplicaciones.

.NET se puede considerar la respuesta de Microsoft al mercado de negocio en entorno Web, como competencia a la plataforma JAVA de Oracle y los framework basados en PHP.

Los principales componentes del marco de trabajo son:

- El conjunto de lenguajes de programación.
- La biblioteca de clases base (BCL).
- El entorno común de ejecución para lenguajes (CLR).

El CLR es el entorno de ejecución donde se cargan las aplicaciones desarrolladas en los distintos lenguajes. CLR permite integrar proyectos en distintos lenguajes soportados por la plataforma. La herramienta de desarrollo que compila el código fuente en código intermedio es CIL (Common Intermediate Language) similar al bytecode de java, para generarlo el compilador se basa en la especificación CLS (Common Language Specification). Para ejecutarse hace falta un segundo paso, un compilador JIT que genera el código máquina que se ejecuta en el cliente, de esta manera .NET consigue la independencia de la plataforma de hardware. Esta compilación la realiza el CLR a medida que invoca métodos.

3.2 Instalación

Una vez se tiene el ordenador con Windows 7 actualizado y Visual Studio 2010 instalado se procederá a la descarga e instalación del SDK. Es muy importante tener en cuenta dos cosas:

- Que el Kinect esté desenchufado.
- Que Visual Studio esté cerrado.
- Que no haya ningún otro driver para Kinect instalado (incluyendo versiones más antiguas).
- Se descargará el paquete correspondiente (32bits o 64 bits) de la página web www.kinectforwindows.org.
- Se siguen los pasos del instalador.

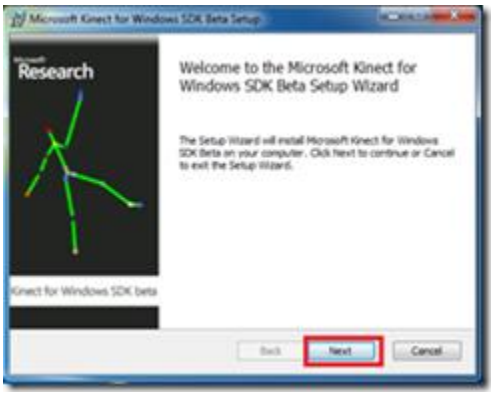


Figura 3.2 Instalador del SDK

Una vez instalado el SDK, se conectará Kinect a la corriente y al ordenador (por USB), el sistema operativo reconocerá el nuevo hardware y se dispondrá a instalarlo automáticamente.

Si se hace click en más información se mostrará el siguiente estado al acabar la instalación:

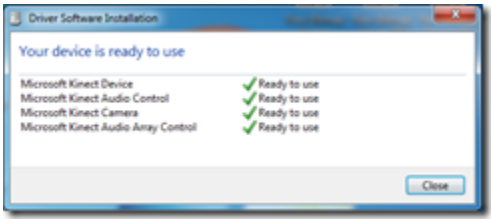


Figura 3.3 Instalación completada

Esta instalación deberá hacerse en cada equipo en donde se quiera usar Kinect, para el desarrollo de aplicaciones así que como para la ejecución de aplicaciones que lo usan.

Para comprobar si se han instalado los drivers de Kinect correctamente, basta con ir a “Panel de control ->Sonido y Hardware ->Administrador de dispositivos” y ahí aparece el sensor instalado:

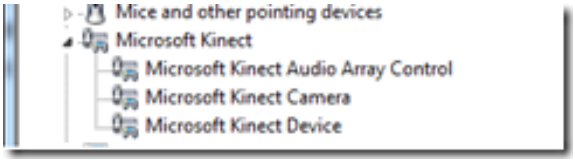


Figura 3.4 Sensores instalados

3.3 Recomendaciones de uso

- Es necesario conectar el dispositivo a una fuente de alimentación externa, ya que sino su funcionalidad será limitada.

- El sistema está protegido frente al sobrecalentamiento, a partir de 90º se apaga automáticamente, no hay ninguna interfaz que permita regular el ventilador. La temperatura ambiente adecuada está entre 5º y 35º

- El motor de inclinación no está preparado para usarse frecuentemente, no es recomendable más de un cambio al segundo o más de quince cambios en un periodo de 20 segundos, esta limitación viene dada con el objetivo de proteger el hardware, si una aplicación intenta usar frecuentemente el motor el sistema primero bloqueará la aplicación y si sigue así devolverá un código de error.

- No colocar nunca Kinect enfrente de un altavoz o sobre una superficie que vibre.

- No colocar nunca Kinect en el exterior o directamente a la luz del sol.

- No colocarlo nunca cerca de una fuente de calor.

- No conectar Kinect en un hub que es usado por otros dispositivos.

- Evitar el uso de otro software o drivers beta que puedan interactuar con el SDK beta de Kinect.

3.4 Rangos para el uso del sensor Kinect[2]

El sensor Kinect fue originalmente diseñado para su uso para Xbox en aplicaciones de salón. Este SDK abre la posibilidad de reutilizar este sensor en otros contextos, pero esta reutilización no cambia el diseño físico ni capacidades del sensor.

[2] Traducido de Programming Guide Microsoft Kinect SDK Beta 1 Draft Version 1.1

Es necesario destacar que el detector de esqueletos sólo detecta esqueletos de figuras que estén de pie, no reconoce figuras sentadas.

Rango de Profundidad	
Sensor de color y profundidad	De 1,2 metros a 3,5 metros
Detección de esqueletos	De 1,2 metros a 3,5 metros

Si el jugador se encuentra muy cerca del sensor los datos obtenidos serán erróneos.

Especificaciones del sensor Kinect	
Angulo de visión	43º de campo de visión vertical 57º de campo de visión horizontal
Angulo de movimiento del motor	+28º y -28º
Velocidad de frames	30 frames por segundo (FPS)
Resolución (profundidad)	QVGA (320 x 240) 16bits
Resolución (Cámara RGB)	VGA (640 x 480) 32 bits
Formato de Audio	16-kHz, 16-bit PCM
Características de entrada de Audio	Cuatro micrófonos en array con un convertidor analógico a digital de 24 bits (ADC) y el procesamiento de señales de Kinect con cancelación del eco y eliminación de ruido de fondo.

3.5 Arquitectura

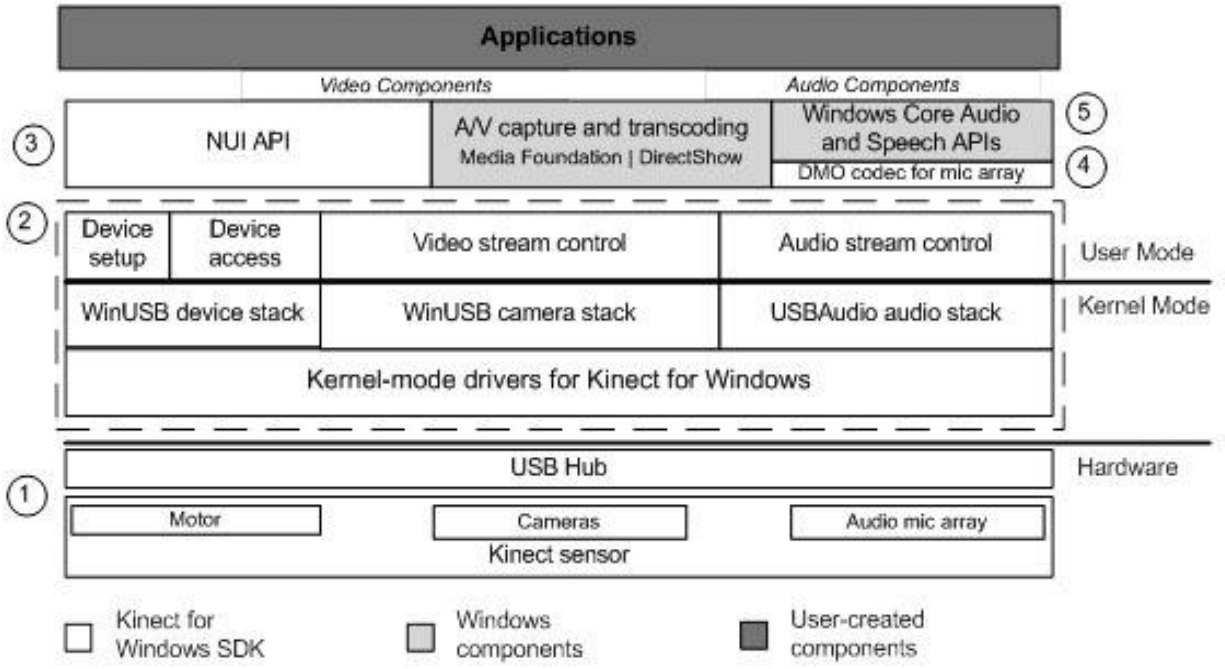


Figura 3.5 Arquitectura de Kinect para Windows[3]

Donde los números hacen referencia a:

1. Hardware Kinect.
2. Los Drivers de Kinect desarrollados para Microsoft Windows 7, son los instalados por el SDK y proporcionan soporte para:
    - El sistema de micrófonos como un dispositivo kernel-mode al que podremos acceder con las APIs de audio estándares de Windows.
    - Streaming de imagen y datos de profundidad.
    - Funciones de enumeración de dispositivos que permitirán a una aplicación utilizar más de un sensor Kinect (conectado a la misma computadora).
3. NUI API, son el conjunto de APIs que recogen los datos capturados por los sensores de imagen y que controlan el dispositivo.
4. Kinect Audio DMO, amplia el soporte de micrófonos de Windows7 para exponer la localización de la formación de la fuente.
5. APIs estándar de Windows.

[3] Obtenida de http://deviak.com/2690172

3.6 NUI API

Esta API es la principal de Kinect para Windows, es la API que da soporte al manejo de datos así como el manejo de algunas de las propiedades del dispositivo:

- Acceso a los diferentes Kinect que están conectados a un dispositivo.
- Acceso a los streams de imágenes y de datos de profundidad captados por los sensores de Kinect.
- Manejo de las imágenes procesadas y datos de profundidad para dar soporte al Skeletal tracking.

Esta API está dividida en cuatro subsistemas, al inicializar la API tendremos que seleccionar cual o cuales de los subsistemas utilizaremos:

- Color, la aplicación envía los datos de la imagen captada por el sensor.
- Depth, envía los datos de profundidad captados por el sensor.
- DepthandPlayerIndex, devuelve los datos de profundidad y si el pixel se corresponde con un jugador devuelve el índice del jugador.
- Skeleton, devuelve los datos del esqueleto.

3.6.1 Datos de la imagen a Color

Existen dos formatos de imagen a color:

- RGB color, proporciona mapas de bits a color de 32 bits lineares con formato X8R8G8B8, dentro del espacio de colores sRGB. Para usarlo es necesario hay que especificar *color* o *color_YUV* cuando se abre el flujo de datos.

- YUV color, proporciona mapas de bits de 16 bit con corrección gamma. Como este flujo usa 16 bits por cada pixel, este formato usa menos memoria y asigna menos buffer de memoria cuando se abre el flujo de datos. Para trabajar con YUV es necesario especificar YUV COLOR cuando se abre el flujo de datos. Esta opción está disponible sólo para resolución 640 x 480 y con una frecuencia de 15 fps.

Ambos formatos usan datos tomados por la misma cámara, por lo que representan la misma imagen.

Kinect envía los datos al PC a través de un cable USB, que ofrece un ancho de banda determinado. La imagen que es captada por el sensor a 1280 x 1024 es comprimida y transformada a RGB antes de la transmisión al runtime, el runtime luego descomprime estos datos antes de entregarlos a la aplicación. El uso de la compresión permite el envío de datos a una tasa de 30fps, pero el algoritmo usado implica una pérdida de fidelidad de la imagen.

3.6.1.1 Funcionamiento de la cámara RGB

Cada imagen está compuesta por un conjunto de píxeles. Cada pixel de la imagen está compuesta por cuatro componentes que representan los valores rojo, verde, azul y otro componente más que se trata del valor de transparencia (alfa), el caso de imágenes RGBa, o un valor vacio si es del tipo RGB.

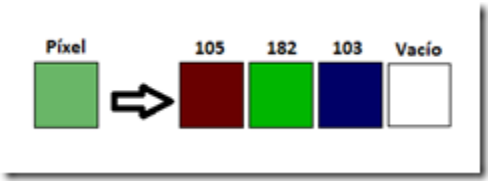


Figura 3.6 Representación de un pixel

Cada componente del pixel tiene un valor entre 0 y 254, que se corresponde a un byte. El sensor Kinect codifica las imágenes que obtiene en un vector de bytes, donde cada byte se corresponde a un componente de cada pixel.

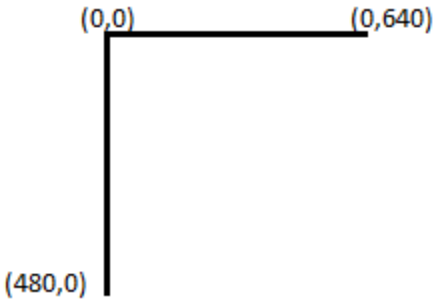


Figura 3.7 Eje de coordenadas imagen RGB

La organización de los pixeles es de arriba a abajo y de izquierda a derecha (véase figura 2.7) donde los cuatro primeros elementos del vector son los valores rojo, verde, azul y alfa del pixel de arriba a la izquierda mientras que los cuatro últimos serán del pixel de abajo a la derecha.

Figura 3.8 Disposición de los pixeles RGB en el array de bytes

3.6.2 Datos de la cámara de profundidad

El flujo de datos de profundidad nos da un frame en el que cada pixel representa la distancia cartesiana entre el sensor y el objeto más cercano en la coordenada x, y del campo de visión del sensor.

El tamaño de los frames puede ser variado:

- 640 x 480
- 320 x 240
- 80 x 60

Sin embargo el SDK sólo admite la resolución 320 x 240.

Al igual que la cámara RGB los datos están codificados en forma de array de bytes donde cada pareja de bytes se corresponde a un pixel, la disposición de los pixeles también es la misma:

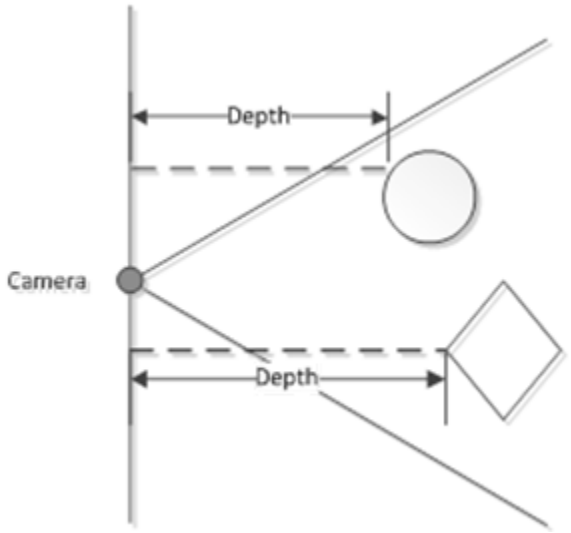


Figura 3.9 Sensor de profundidad

El hecho que cada pixel se corresponda con bytes se debe a que como Kinect tiene dos cámaras de infrarrojos cada pixel guarda la distancia de una cámara al objeto.

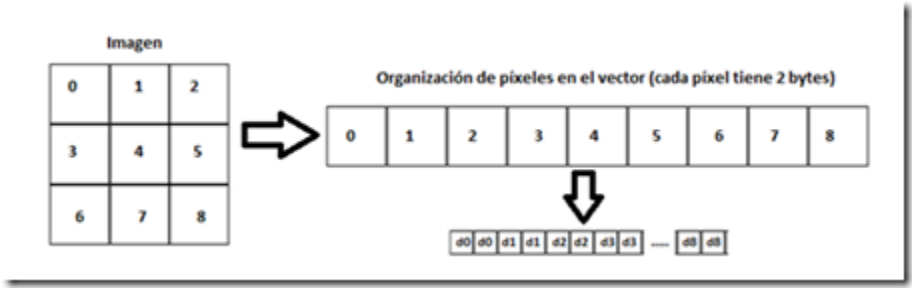


Figura 3.10 Disposición de los pixeles de profundidad en el array de bytes

El formato de los datos de profundidad depende de si se ha especificado el modo Depth o el modo DepthandPlayerIndex.

- Si se ha seleccionado el modo Depth los datos de profundidad están en los primeros 12 bits de cada pixel, estando los cuatro últimos bits no usados.
- Si se ha elegido Depth and player index, los primeros tres bits del pixel contienen el índice de cada jugador, siendo cero en caso de que ese pixel no pertenezca a ningún jugador, aunque usa tres bits, el SDK sólo es capaz de reconocer dos jugadores.

Un valor de profundidad igual a cero indica que no hay datos de profundidad disponibles para esa posición, puede ser debido a que todos los objetos están o muy cerca de la cámara o muy lejos de ella.

3.6.3 Skeletal Tracking

Es la funcionalidad estrella de Kinect, con esto se puede hacer un seguimiento del esqueleto del jugador y se basa en un algoritmo que identifica partes del cuerpo de las personas que están en el campo de visión del sensor. Por medio de este algoritmo se pueden obtener puntos que hacen referencia a las partes del cuerpo de una persona. El esqueleto representa la posición y postura actual de un máximo de dos jugadores.

El Skeletal tracking da la información del esqueleto en forma de un conjunto de puntos (Joints) que componen el esqueleto (ver figura 2.11).

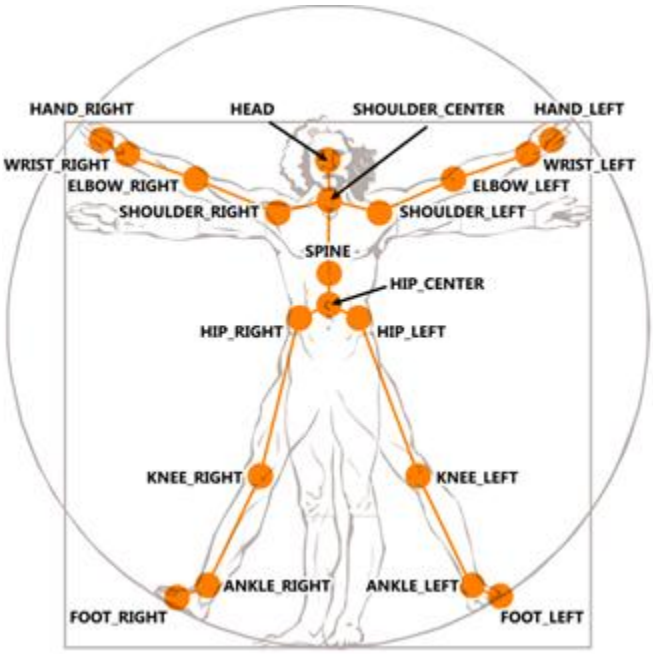


Figura 3.11 Joint detectados por Kinect

El Skeletal tracking crea un Skeleton Frame cada vez procesa los datos de profundidad, aparezca o no un esqueleto en el frame. El esqueleto devuelto contiene el timestamp de la correspondiente imagen de

profundidades de manera que las aplicaciones pueden unir los datos del esqueleto con su imagen de profundidades.

Como se ha dicho el Skeleton Engine es capaz de detectar dos esqueletos completos de dos jugadores que estén en el campo de visión del sensor, es lo que se conoce como un tracking activo. También existe un tracking pasivo (se hace automáticamente) para hasta otros cuatro jugadores que estén en el campo de visión del Kinect. Cuando el tracking es pasivo sólo se almacenará de ellos información acerca de su posición. Por defecto los esqueletos que el sistema asignará tracking activo serán los dos primeros en ser detectados.

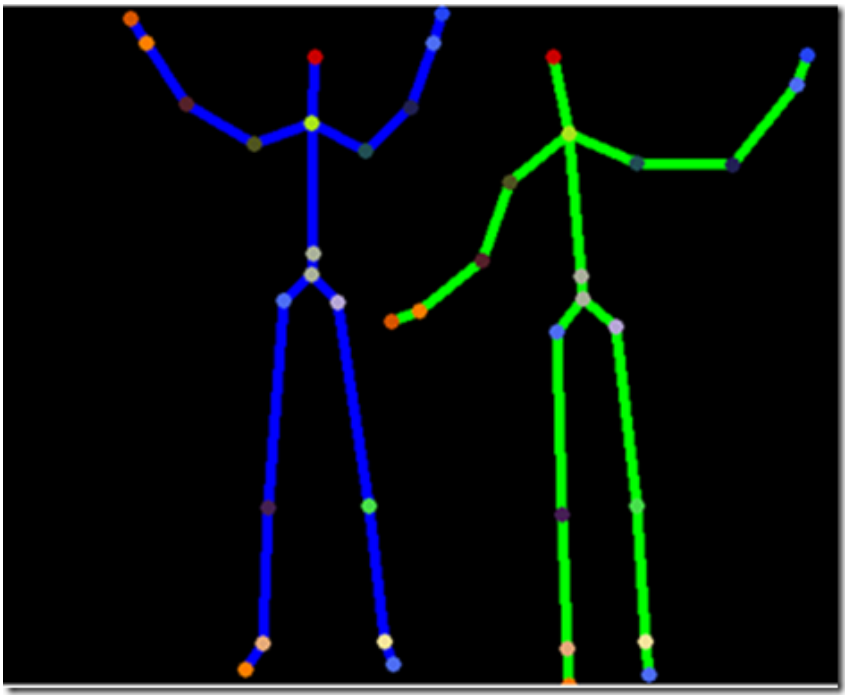


Figura 3.12 Esqueletos con tracking activo

Un SkeletonFrame está compuesto por un array de SkeletonData (clase que contiene los datos de cada esqueleto), uno por cada esqueleto que el Skeleton tracking reconoce. No todos los Skeleton Frame contienen SkeletonData.

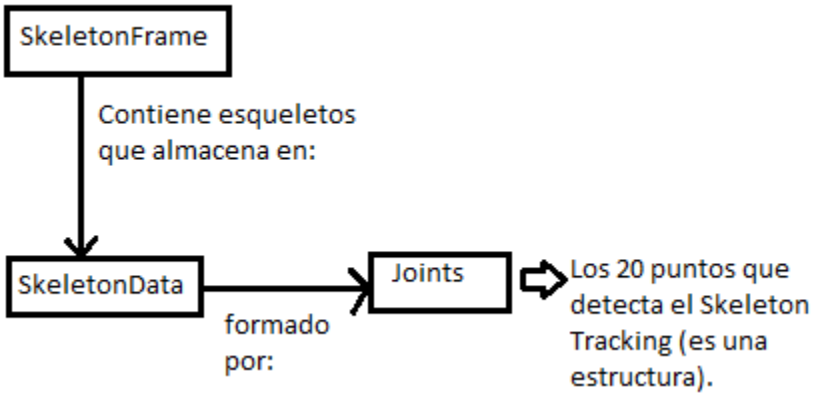


Figura 3.13 Esquema Skeleton Tracking

Para todos los esqueletos detectados, tenemos los siguientes datos:

- El tracking state de los esqueletos es:
  - Tracked para los activos.
  - Position-only para los demás.
- A cada jugador le corresponde un único ID mientras se mueva dentro del rango de visión del sensor. Si sale del campo de visión de Kinect es posible que pierda su ID.
- El valor de la posición del jugador (el centro de masa del jugador), este valor es el único valor que tienen los jugadores con tracking pasivo.

La posición de cada Joint viene determinada por coordenadas x, y y z, a diferencia de los datos de la imagen de profundidad, las medidas están expresadas en metros.

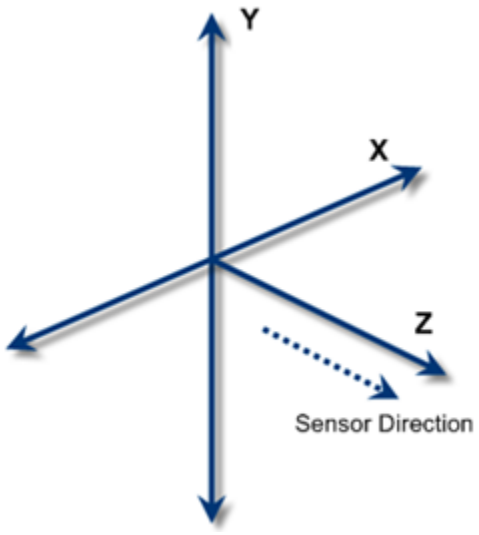


Figura 3.14 Eje de coordenadas esqueleto

Como se puede ver, se trata de un sistema diestro de coordenadas que coloca el conjunto de sensores en el punto de origen con el lado positivo del eje z se extendiéndose en la dirección en que apunta el conjunto de sensores, el lado positivo del eje y extendiéndose hacia arriba y el lado positivo del eje x extendiéndose hacia la izquierda (respecto el Kinect). Cada eje indica:

- Eje X, Posición horizontal, viene dada por la distancia en metros del Kinect al Joint a lo largo del eje X.
- Eje Y, Posición vertical, viene dada por la distancia en metros del Kinect al Joint a lo largo del eje Y.
- Eje Z, Distancia desde el Kinect medida en metros.

3.6.4 Recuperación de información

La aplicación obtiene el último frame (de imagen, Skeleton o Depth) llamando a un método de recuperación de frames pasándole un buffer. Una vez el frame es obtenido, este es copiado en el buffer. Si el código reclama frames es más rápido de lo que estos están disponibles se puede elegir entre esperar a que se genere el siguiente frame o volver a intentarlo otra vez. NUI sólo devuelve el mismo frame una sola vez.

Existen dos métodos para acceder a los datos, no se pueden utilizar estos métodos simultáneamente.

- Polling mode, es la opción más simple para leer frames, se abre el flujo de datos y se especifica cuanto tiempo se quiere esperar por el siguiente frame (entre 0 e infinitos milisegundos). El método de petición envía una respuesta cuando se produce un nuevo frame o cuando se acaba el tiempo, lo que suceda antes. Si la respuesta del método es positiva, significará que se ha generado un nuevo frame. Hay que hacer dos consideraciones en cuanto al tiempo de espera:
    - Poniendo el tiempo a infinito implicará que la llamada al método de petición bloqueará y esperará todo el tiempo que sea necesario para que se produzca el siguiente frame.
    - Si el tiempo de espera se pone a cero, el código de la aplicación puede ir comprobando si se ha producido un nuevo frame mientras se lleva a cabo otra tarea en el mismo hilo.
- Event mode, este modo permite la recuperación de información de una manera mucho más eficaz. Mediante manejadores de eventos cada vez que se genera un frame se llama al manejador de eventos y accede al nuevo frame (y se ejecutan los métodos asociados a ese evento).

3.7 Audio API

El sistema de micrófonos consiste en un conjunto de cuatro micrófonos dispuestos en forma de L. El tener el conjunto de micrófonos en esta posición da diversas ventajas respecto de un micrófono sólo:

- Mejora de la calidad de audio, los conjuntos de micrófonos puedes soportar algoritmos más sofisticados de eliminación de ruido y eco que un micrófono sólo.
- Como el audio llega en diferente tiempo a cada micrófono, se puede determinar la dirección donde está el origen del sonido y usar el conjunto de micrófonos como un micrófono direccional orientable.

Las aplicaciones que usen el SDK podrán utilizar los micrófonos para lo siguiente:

- Captura de audio de alta calidad
- Formación de haz y localización de la fuente (el MSRKinectAudio DMO incluye algoritmos de construcción que controlan el “haz” y proporcionan la dirección del origen a las aplicaciones).
- Reconocimiento de voz

3.8 OpenNI

Aparte del SDK oficial de Microsoft existen otros SDKs no oficiales, OpenNI (open natural interaction) de PrimeSense es el ejemplo de uno de ellos.

OpenNi es un framework que proporciona un conjunto de APIs open source para el manejo de Kinect. OpenNI fue creado en el mes de noviembre de 2010 y su sitio web estuvo disponible a partir del ocho de diciembre de ese mismo año, detrás de OpenNI está la empresa PrimeSense que es una de las que desarrollo la tecnología usada en Kinect. Estas APIs ofrecen soporte para:

- Reconocimiento de voz
- Reconocimiento de gestos de manos
- Reconocimiento de esqueleto

3.8.1 Kinect SDK vs OpenNI

Microsoft Kinect SDK BETA 2	**OPENNI**

Pros:
• Soporte para audio
• Soporta manejo del motor de inclinación
• Tracking de todo el cuerpo
  ○ No necesita calibración previa
  ○ Incluye cabeza, manos pies y clavícula
  ○ Funciona mejor con los Joints ocultos
• Puede manejar varios sensores
• Instalación sencilla
• El SDK incluye eventos que saltan cuanto un nuevo Frame está disponible
	Contras:
	• No soporta Audio
	• No tiene soporte para el motor de inclinación
	• Reconocimiento corporal:
	  ○ Carece de rotaciones para cabeza, manos pies y clavícula
	  ○ Necesita calibrarse antes de empezar el tracking (aunque puede ser grabada y cargada de un disco)
	  ○ Articulaciones no detectadas no se estiman
	• Soporta varios sensores aunque su instalación y enumeración es extraño
	• Tiene tres instaladores separados y una clave de licencia NITE
	No tiene eventos que detecten cuando un nuevo frame de video o profundidad son detectados

Contras: • Licencia no comercial • Solo hace tracking del cuerpo completo • No ofrece alineamiento entre los flujos de color y profundidad con otro • Aunque hay características para	**Pros:** • Con licencia comercial • Incluye un framework para el tracking de manos • Incluye un framework para el reconocimiento de gestos con las manos

alinear coordenadas individuales
• Reconocimiento corporal:
  ○ Solo reconoce las posiciones de los Joints, no las rotaciones
  ○ Reconoce el cuerpo completo, no hay modo para que reconozca la parte de arriba o sólo las manos
  ○ Parece que consume más CPU que OpenNI
• No tiene sistema de reconocimiento de gestos
• No da soporte a otros sensores similares a Kinect
• Sólo soporta Windows 7
• No hay soporte para el motor de juego Unity3D
• No da soporte para la grabación o reproducción desde disco
• No da apoyo para transmitir en bruto de video infrarrojo
• No tiene eventos que indiquen cuando un nuevo jugador entra o sale del frame
	• Alinea automáticamente el flujo de la imagen de profundidad con el de imagen a color
	• Tracking de todo el cuerpo
	  ○ Calcula la rotación de los Joints
	  ○ Soporte de modo sólo manos
	  ○ Parece que consume menos CPU
	• Soporta el sensor de PrimeSense y ASUS, el WAVI Xtion sensors
	• Soporta varios sensores aunque su instalación y enumeración es extraño
	• Soporta Windows (también Vista y XP), Linux y MAC OSX
	• Viene con código de apoyo en el motor de juego Unity3D
	• Soporta la grabación/reproducción desde disco
	• Apoyo para transmitir en bruto de video infrarrojo
	Tiene eventos que indican cuando un jugador entra o sale del frame

En definitiva parece que el SDK de Microsoft es más adecuado cuando se trabaja con esqueletos y Audio y OpenNI parece mejor cuando se trabaja en plataformas que no sean Windows 7 o para proyectos comerciales.

En cuanto al tema del reconocimiento de gestos, si el sensor va a ver solamente la parte alta del cuerpo o las manos, entonces OpenNI es más adecuado porque tiene un framework de reconocimiento gestual. Sin embargo si es el cuerpo entero el que ve el sensor, es más recomendable usar el SDK de Microsoft ya que el reconocimiento de esqueletos es más estable y el framework de reconocimiento de gestos que tiene OpenNI ya no valdría por lo que tendríamos que desarrollarlo desde cero, al igual que con el SDK de Microsoft.

Capítulo 4 Primeras mini-aplicaciones

Con el objetivo de comenzar a familiarizarse con el SDK de Kinect se procederá a realizar pequeñas aplicaciones que sirvan de ejemplo para ver las posibilidades de Kinect así como para coger destreza con el manejo del SDK de Microsoft.

Para la implementación de estas mini-aplicaciones se utilizará la tecnología Microsoft Windows Presentation Foundation (WPF).

4.1 Windows Presentation Foundation

Windows Presentation Foundation es una tecnología Microsoft, presentada como parte de Windows Vista. Permite el desarrollo de interfaces de interacción en Windows.

WPF ofrece una amplia infraestructura y potencia gráfica con la que es posible desarrollar aplicaciones visualmente atractivas, con facilidades de integración que incluyen animación, video, audio, documentos, navegación o gráficos 3D. Cabe destacar que separa con el lenguaje declarativo XAML y los lenguajes de programación .NET, la interfaz de la lógica del negocio, propiciando una arquitectura Modelo Vista Controlador.

WPF ofrece interoperabilidad con el API de Windows, se puede utilizar WPF dentro del código existente de WIn32 o viceversa. También ofrece interoperabilidad con los formularios Windows.

WPF se incluye con Windows 7, Windows Vista y Windows Server 2008 aunque también está disponible para Windows XP SP2 o posterior y Windows Server 2003.

4.1.1 XAML

WPF introduce un nuevo lenguaje conocido como lenguaje extensible de marcado de aplicaciones, XAML, basado en XML. Su objetivo es dar un método más eficaz de desarrollo de interfaces de usuario.

Su gran ventaja es que es un lenguaje completamente declarativo, de manera que el desarrollador puede describir el comportamiento y la integración de los componentes sin utilizar la programación procedural, aunque es raro que una aplicación se defina completamente en XAML. Su utilización para desarrollar interfaces de usuario permite también la separación entre modelo y vista, lo que es un buen principio de arquitectura. En XAML, los elementos y atributos mapean las clases y propiedades del API subyacente.

Hay que destacar que el uso de XAML en el desarrollo de aplicaciones no es estrictamente necesario, es posible cargar todos los elementos de WPF mediante código (C#, VISUAL Basic.NET. Sin embargo XAML es más rápido, más fácil de implementar y de localizar y significa una elección mejor que cualquier código equivalente. Con XAML no existen errores de rendimiento, ya que es una representación de un modelo de objetos basado en XML.

4.2 Preparación del Proyecto

Para comenzar un proyecto se empezará creando un proyecto WPF. Una vez se asigna un nombre y se acepta se creará un nuevo proyecto. Una vez creado se tienen dos ficheros:

- MainWindow.xaml, en este fichero de diseño será donde se trabaje la parte gráfica (la interfaz) de la aplicación.
- MainWindow.xaml.cs, en esta parte se encontrará la lógica de negocio de la aplicación.

Lo primero que se debe de hacer es agregar la referencia al SDK Kinect, para ello se va al explorador de proyectos y se hace clic encima de References y se selecciona "agregar referencia" y una vez ahí se busca "Microsoft.Research.Kinect" y se acepta, una vez hecho esto la referencia aparecerá en el explorador de proyectos.

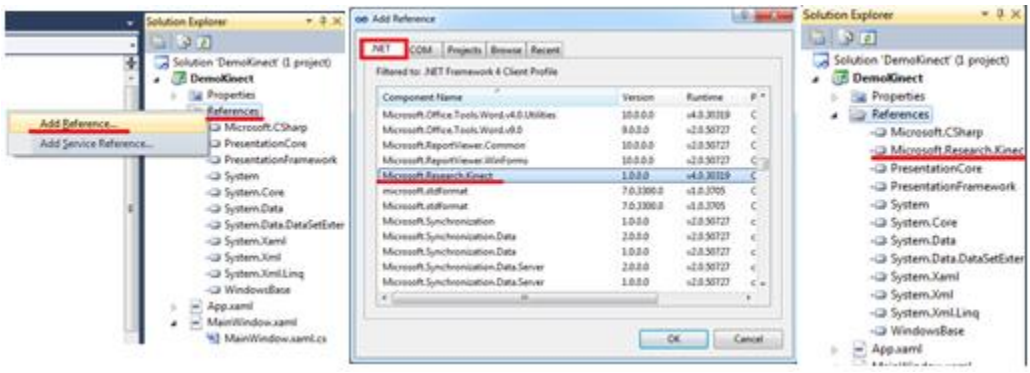


Figura 4.1 Agregación referencia al SDK

Una vez agregada la referencia es necesario agregarla al espacio de nombres para poder usar sus clases y sus métodos, para ello hay que añadir un using más:

"using Microsoft.Research.Kinect.Nui"

Ahora se tendrá que añadir unas líneas de código en el constructor de la clase (método MainWindow()), estas líneas de código son eventos que están destinados a señalar los métodos que deben ejecutarse cuando se cargue la interfaz y cuando se cierre la aplicación:

```
public MainWindow()

{

InitializeComponent();

Loaded += new RoutedEventHandler(MainWindow_Loaded);

Closed += new EventHandler(MainWindow_Closed);

}
```

Figura 4.2 Constructor de una aplicación para Kinect

Los métodos MainWindow_Loaded y MainWindow_Closed son los métodos que saltarán cuando se cargue la pantalla y cuando se cierre respectivamente.

En el método MainWindow_Loaded será donde se instancie la clase que hará referencia al sensor y donde abriremos los flujos que utilizaremos en la aplicación (imagen, profundidad, Skeleton tracking…).

La clase que representa al sensor es la clase Runtime, cuando se instancie se tendrá que introducir como opciones una o varias de las cuatro opciones que tiene el Kinect:

- Color
- Depth
- DepthandPlayerIndex
- UseSkeletalTracking

```
void MainWindow_Loaded(object sender, RoutedEventArgs e)

{

kinect = new Runtime();

kinect.Initialize(RuntimeOptions.UseColor |
RuntimeOptions.UseDepthAndPlayerIndex |
RuntimeOptions.UseSkeletalTracking);

}
```

Figura 4.3 Ejemplo de MainWindow_Loaded

Por su parte el método MainWindow_Closed se encarga de la de la finalización del sensor cuando se cierre la aplicación.

```
void MainWindow_Closed(object sender, EventArgs e)

{

kinect.Uninitialize();

}
```

Figura 4.4 Ejemplo de MainWindow_Closed

Esta es la estructura básica a partir de la cual se empezará a desarrollar el resto de aplicaciones con Windows Presentation Foundation.

4.3 Primera Mini-aplicación: Uso de cámara RGB y profundidad

El objetivo de esta mini-aplicación que tiene como objetivo el empezar a familiarizarse con el empleo de las cámaras, la aplicación mostrará dos imágenes, una con la imagen a de lo que está viendo el sensor y otra esta imagen donde se pintarán los pixeles según el rango de distancia en el que se encuentren:

- Azul, si la distancia es más cercana a 900 mm
- Verde, si está entre 900 y 2000 mm
- Roja, si está a más de 2 metros

También se deberá mostrar en un color diferente los pixeles correspondientes a un jugador.

Para mostrar los frames captados por el sensor se dibujaran dos elementos "image" en el XAML, para hacerlo se puede hacer uso del editor gráfico, en código cada elemento tiene esta estructura:

"<Image Height="282" HorizontalAlignment="Left" Margin="20,12,0,0" Name="image1" Stretch="Fill" VerticalAlignment="Top" Width="312" />"

Una vez hecho esto se pasa a la apertura de los flujos de color y profundidad, estos flujos se abren dentro del método MainWindow_Loaded:

```
void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    kinect = new Runtime();
    kinect.Initialize(RuntimeOptions.UseColor | RuntimeOptions.UseDepthAndPlayerIndex | RuntimeOptions.UseSkeletalTracking);

    kinect.VideoStream.Open(ImageStreamType.Video, 2, ImageResolution.Resolution640x480, ImageType.Color);
    kinect.VideoFrameReady += new EventHandler<ImageFrameReadyEventArgs>(kinect_VideoFrameReady);

    kinect.DepthStream.Open(ImageStreamType.Depth, 2, ImageResolution.Resolution320x240, ImageType.DepthAndPlayerIndex);
    kinect.DepthFrameReady += new EventHandler<ImageFrameReadyEventArgs>(kinect_DepthFrameReady);
}
```

Figura 4.5 Apertura de flujos de imagen y profundidad

Los métodos "open" permiten la apertura del flujo de la cámara de video y de la cámara de profundidad, este método recibe una serie de parámetros:

- En primer lugar se indica el tipo de flujo que se abre (Video y Depth aunque también existe el tipo Invalid)
- El poolsize, normalmente es 2.
- La resolución, debe de ser recomendada con el tipo de imagen con la que se está tratando (ver punto 2.4)
- El tipo de imagen, en este caso DepthandPlayerIndex y Color.

Después para cada flujo se crea un evento para capturar el frame que devuelve el flujo de la cámara cuando este está listo, momento en el que se ejecutarán los métodos "kinect_VideoFrameReady()" y "kinect_depthFrameReady" donde se obtendrá del frame la imagen codificada y se trabajará con ella.

4.3.1 Tratamiento de la imagen RGB

El tratamiento de la imagen se realiza en el método "Kinect_VideoFrameReady" que es llamado cada vez que se genera un nuevo frame. A partir de este nuevo frame es posible obtener la imagen captada por el sensor, esta imagen está codificada dentro de la estructura de datos PlanarImage.

A partir de esta imagen se creará un BitmapSource que será la imagen que se mostrará en el elemento Image creado anteriormente en el fichero XAML, es decir que cada vez que se cree un frame ira actualizando el elemento Image.

Para crear el BitmapSource se usa el método create() de la clase, en este método se le pasan los siguientes parámetros:

- Anchura del PlanarImage (en píxeles)
- Altura del PlanarImage (en píxeles)
- dpiX (puntos por pulgada en el ejeX, es independiente de la pantalla), en WPF es 96.
- dxiY (puntos por pulgada en el ejeY), 96
- El formato de los pixeles, en este caso se usa bgr32
- Palette, se usa null
- La imagen en bits capturada por el sensor, es un array de bytes
- El stride, es el numero de bytes a lo ancho de la imagen, en este caso es el ancho del PlanarImage (en píxeles) multiplicado por cuatro debido a que cada pixel de la cámara de color lo forman cuatro bytes (ver punto 2.6.1.1).

```
void kinect_DepthFrameReady(object sender, ImageFrameReadyEventArgs e)
{
    PlanarImage image = e.ImageFrame.Image;
    byte[] convertedFrame = convertDephFrame(image.Bits);
    image2.Source = BitmapSource.Create(image.Width, image.Height, 96, 96, PixelFormats.Bgr32, null, convertedFrame, image.Width * 4);

}
```

Figura 4.6 Tratamiento de frame de imagen RGB

4.3.2 Tratamiento de la imagen de profundidad

Al igual que con la imagen a color se obtiene la imagen codificada a partir del frame. Como ya se habló en el punto 2.6.2 en esta imagen cada pixel está formado por dos bytes que indican a la distancia a la que se encuentra ese pixel.

Para mostrar la imagen como se quiere, es decir si se quiere que cada pixel muestre un color según el rango de distancia en el que está, se deberá crear un array de bytes que represente una imagen RGB que se construirá a partir de la imagen que se obtiene del frame.

```
private byte[] convertDephFrame(byte[] dephFrame16)
{
    int distance;
    byte[] dephFrame32 = new byte[320 * 240 * 4];
    for (int i16 = 0, i32 = 0; i16 < dephFrame16.Length && i32 < dephFrame32.Length; i16 += 2, i32 += 4)
    {
        distance = (dephFrame16[i16] >>3 | dephFrame16[i16 + 1] << 5);
        if (distance <= 900)
        {
            dephFrame32[i32 + Red_idx] = 0;
            dephFrame32[i32 + Green_idx] = 0;
            dephFrame32[i32 + Blue_idx] = 255;
        }
```

Figura 4.7 Tratamiento datos profundidad

Como se puede ver en la Figura 3.7 para realizar el tratamiento de los datos de profundidad se llama aun nuevo método donde se irá recorriendo el array de bytes de profundidad. Se creará un nuevo array donde se guardará el resultado del recorrido, el tamaño de este array será la resolución de la imagen obtenida del frame (como es de profundidad será 320 x 240) multiplicada por el número de bytes por pixel (como se quiere obtener una imagen RGB será cuatro).

Los bytes de este array se irán rellenando según los valores que obtengamos de los datos de profundidad, por ejemplo, como se puede ver en la figura 3.7, si la distancia es menor que 900mm el color con el que se dibujará ese pixel es azul.

Para calcular la distancia es necesario hacer una operación OR entre los dos bytes, haciendo OR de los bits más significativos con los más significativos y así sucesivamente. Sin embargo al haber seleccionado la opción DepthandPlayerIndex los primeros tres bits del primer byte indican el identificador del jugador, por lo que hay que hacer la operación indicada en la figura 3.8.

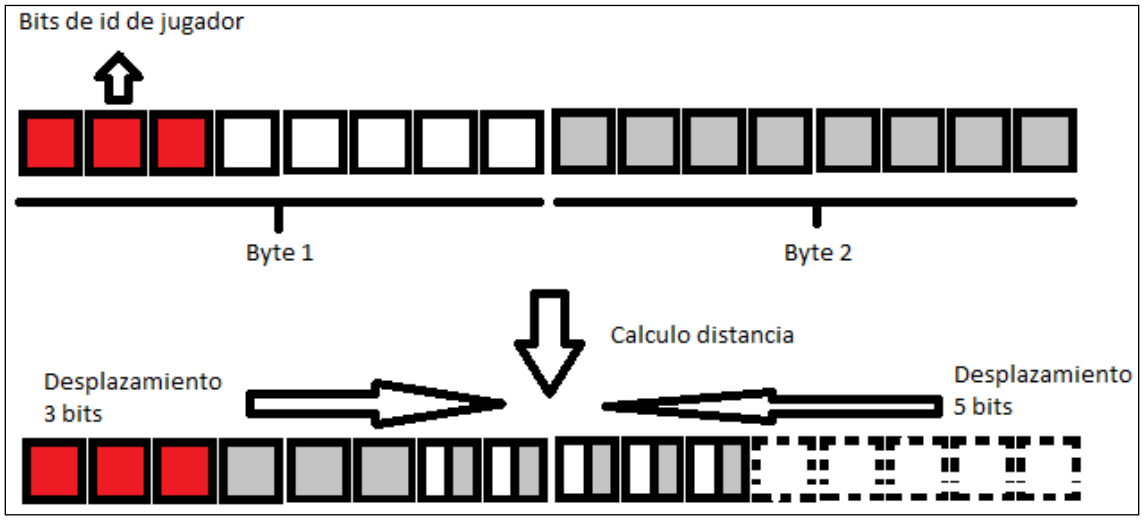


Figura 4.8 Calculo de la distancia

Como se puede ver en la figura en vez de hacer un desplazamiento de todo el segundo byte, lo que se hace es superponer los cinco bits menos significativos del primer byte con los cinco menos significativos del segundo byte y haciendo una operación OR.

Por último, como ya se había dicho en los requisitos, es necesario comprobar si el pixel seleccionado pertenece a un jugador, para eso se comprobarán los primeros tres bits del primer byte y si es distinto que cero esto indicará que se trata de un jugador por lo que se coloreará de un color distinto (ver figura 3.9).

```
int player = dephFrame16[i16] & 0x07;
if (player != 0)
{
    dephFrame32[i32 + Red_idx] = 0;
    dephFrame32[i32 + Green_idx] = 0;
    dephFrame32[i32 + Blue_idx] = 0;
}
```

Figura 4.9 Comprobación índice jugador

El array de bytes resultante de este método se mostrara en el elemento imagen correspondiente a través de un BitmapSource de manera similar a como se realizó para imágenes a color en el punto anterior.

El resultado de esta mini-aplicación es el siguiente:

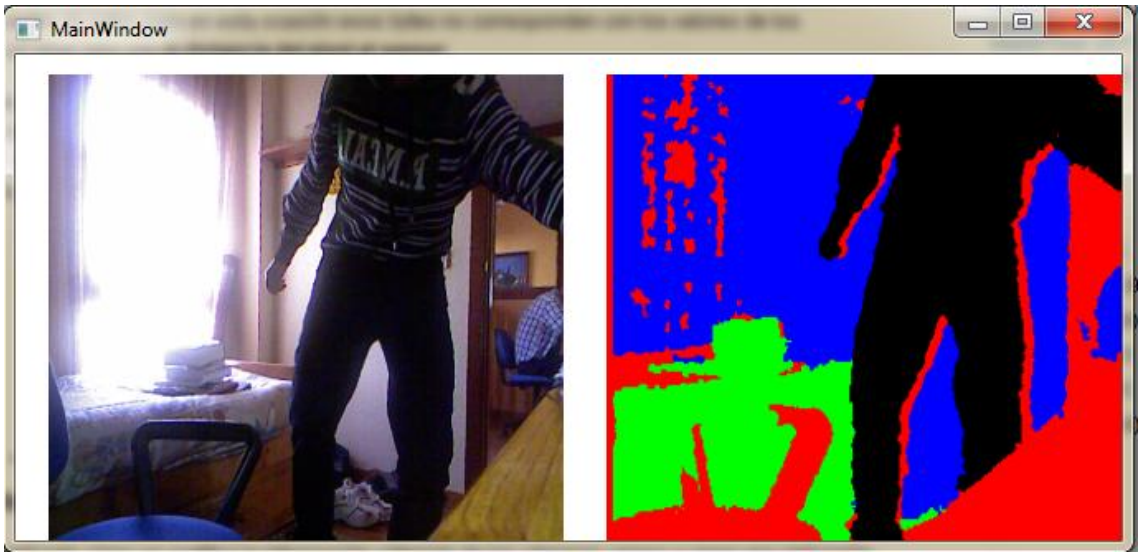


Figura 4.10 Resultado mini-aplicación 1

Como se puede observar algunos objetos que están muy próximos los pone como lejanos (color rojo) esto se debe a que están demasiado cerca de Kinect (menos que 1,2 metros) lo que hace que las lecturas del sensor sean erróneas.

4.4 Segunda Mini-aplicación: Skeletal Tracking

El objetivo de esta mini-aplicación es familiarizarse con el Skeletal tracking y algunas de las posibilidades que ofrece. Esta aplicación mostrará tres imágenes:

- La imagen a color captada por el sensor
- La imagen con los rangos de profundidad
- El esqueleto dibujado de un jugador

Para las dos primeras se usará el código desarrollado en la anterior mini-aplicación, por lo que no se volverá a explicar.

Además de mostrar el dibujo del esqueleto, la aplicación deberá reconocer unas posturas y gestos determinados haciendo uso de la información que da el esqueleto del jugador.

Como se dijo en el capítulo anterior, el SDK de Kinect tiene implementada la detección de gestos y posturas por lo que será necesario implementarlo. Para facilitar su posterior reutilización tanto las clases que implementan los detectores como los métodos que ayudan al dibujo del esqueleto serán desarrolladas en clases aparte.

La estructura final de clases queda representada en el diagrama de clases de la figura 3.11.

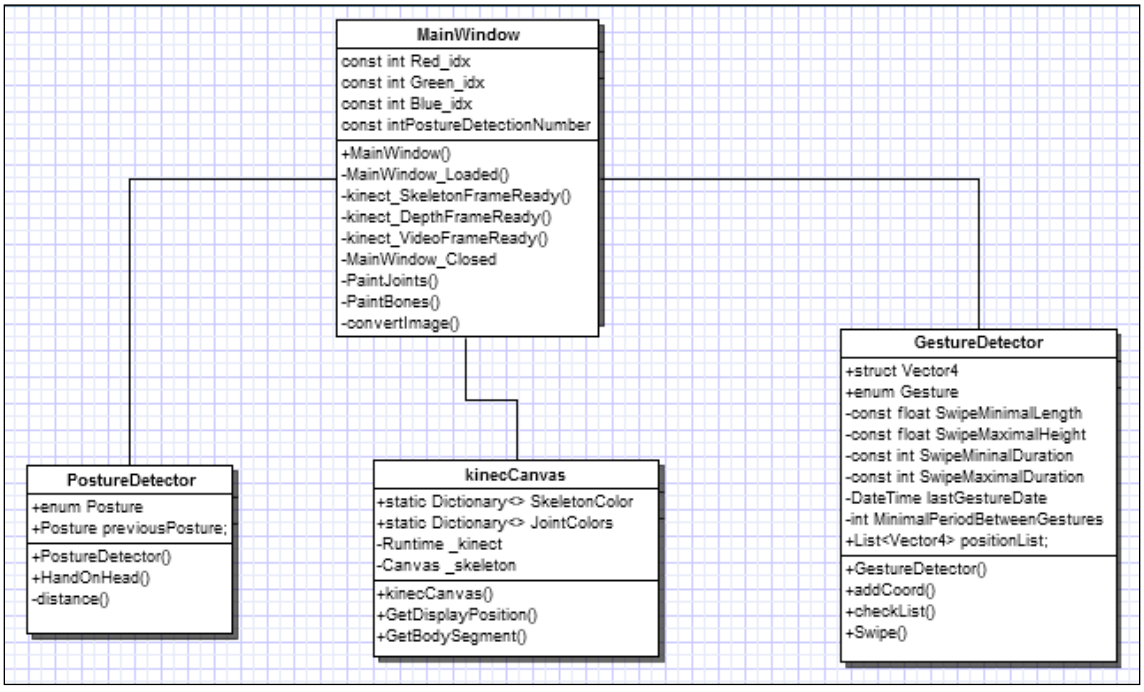


Figura 4.11 Diagrama de Clases mini-aplicación 2

Debido a la naturaleza del lugar de trabajo, para que todo el cuerpo del jugador este en el campo de visión del sensor y así facilitar el reconocimiento del esqueleto, se va a hacer uso del motor para subir la inclinación 10º.

Para ello en el método MainWindow_Loaded() después de inicializar el sensor se añadirá la siguiente línea de código:

"kinect.NuiCamera.ElevationAngle = 10;"

Una vez se cierre la aplicación para que el sensor vuelva a su posición habitual es necesario que en el método MainWindow_Closed() se añada la siguiente línea de código (esta línea tiene estar antes de la finalización del sensor).

"kinect.NuiCamera.ElevationAngle = 0;"

4.4.1 Lectura de Joints y dibujo del esqueleto

Lo primero de todo que hay que hacer si se quiere trabajar con el Skeletal Tracking es añadir la opción "*RuntimeOptions.UseSkeletalTracking"* cuando se inicializa el sensor (con el método Initialize().

Una vez que se ha inicializado el sensor es necesario asociar al evento SkeletonFrameReady a un método para que este se ejecute cada vez que se tenga un frame nuevo, es este caso el método es *"kinect_SkeletonFrameReady"*.

```
void kinect_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
    int skeletonId = 0;
    canvas1.Children.Clear();
    foreach (SkeletonData skeleton in e.SkeletonFrame.Skeletons)
    {
        if (skeleton.TrackingState == SkeletonTrackingState.Tracked)
        {
            //kinec solo puede detectar dos esqueletos
            PaintBones(skeleton,kinecCanvas.SkeletonColor[skeletonId]);
            PaintJoints(skeleton);
```

Figura 4.12 kinect_SkeletonFrameReady

Como se puede ver en el código de la figura 3.12, dentro de un mismo frame pueden aparecer varios esqueletos (hasta 7) pero sólo interesa trabajar con aquellos que están activos (aquellos que tienen un tracking activo, ver punto 2.6.3), estos Skeleton tienen la información de la posición de 20 puntos del cuerpo (Joints), estos puntos son los que se dibujaran y se corresponderán con las articulaciones del esqueleto, una vez dibujados los puntos se dibujarán líneas uniendo algunos de estos puntos y es lo que dará lugar al esqueleto.

4.4.1.1 Dibujo del esqueleto

Lo primero de todo es determinar dónde se va a dibujar el esqueleto, en el XAML se creará un elemento Canvas de la misma manera que se creó el elemento Image, el color de fondo del Canvas será negro y se vaciará cada vez que se lea un nuevo frame con:

"Canvas.children.Clear();"

Para dibujar el esqueleto se ha creado la clase kinecCanvas donde se encuentran métodos que ayudan a dibujar en el elemento Canvas y que se podrá reutilizar e otras aplicaciones:

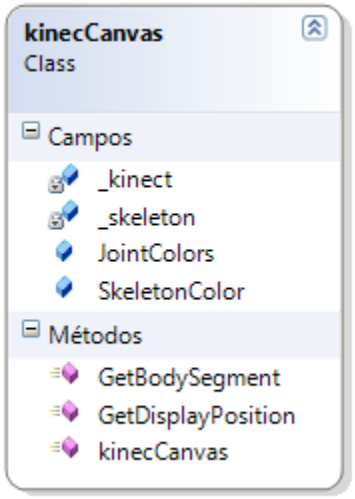


Figura 4.13 Clase kinecCanvas

Como se puede ver en la figura 3.13 los campos de la clase kinecCanvas son los siguientes:

- _kinect, es una referencia al sensor Kinect, se introduce en el constructor
- _skeleton, es una referencia al Canvas donde se dibujará el Skeleton, hay que introducirlo como parámetro en el contructor.
- JointColors, es un diccionario que da a partir del ID del Joint el color con el que se tiene que pintar ese Joint.
- SkeletonColor, para cuando hay más de un esqueleto indica de qué color hay que pintar las líneas que unen los Joints de cada esqueleto.

Los métodos de la clase son los siguientes:

- kinecCanvas, es el constructor, toma como variables de entradas la referencia al sensor y al elemento Canvas donde se va a dibujar el esqueleto.
- GetDisplayPosition, este método a partir de un Joint hace una transformación de sus coordenadas y devuelve una estructura System.Windows.Point con las coordenadas del Joint adaptadas al eje de coordenadas y al tamaño del elemento Canvas.
  Para hacer esto se hace uso de los métodos de SDK:

- SkeletonToDepthImage, que pasa las coordenadas del Joint a un eje de coordenadas como el de las imágenes de profundidad (en dos dimensiones), luego estas coordenadas x, y hay que pasarlas a una resolución de 320 x 240.
- GetColorPixelCoordinatesFromDepthPixel, que devuelve las coordenadas del pixel que corresponde al pixel de profundidad en un eje de coordenadas como el de las imágenes a color.

Después de aplicar el último método, se obtiene unas coordenadas que se corresponden a un eje con una resolución de 640 x 480 así que habrá que adaptar estas coordenadas al tamaño del Canvas en el que se va a dibujar el esqueleto.

```
public Point GetDisplayPosition(Joint joint)
{
    float depthX, depthY;
    _kinect.SkeletonEngine.SkeletonToDepthImage(joint.Position, out depthX, out depthY);
    depthX = Math.Max(0, Math.Min(depthX * 320, 320));  //convert to 320, 240 space
    depthY = Math.Max(0, Math.Min(depthY * 240, 240));  //convert to 320, 240 space
    int colorX, colorY;
    var iv = new ImageViewArea();
    _kinect.NuiCamera.GetColorPixelCoordinatesFromDepthPixel(ImageResolution.Resolution640x480, iv, (int)depthX, (int)depthY,
    return new Point((int)(_skeleton.ActualWidth * colorX / 640.0), (int)(_skeleton.ActualHeight * colorY / 480));
}
```

Figura 4.14 Método GetDisplayPosition()

- GetBodySegment, este método unirá una serie Joints mediante líneas y devolverá un elemento polyline (clase que contiene varias líneas). Antes de crear el método Polyline será necesario llamar al método GetDisplayPosition() para cada Joint.

Una vez explicada la clase KinecCanvas veamos cómo funciona el dibujo del esqueleto en la aplicación.

Lo primero que se hace es dibujar los 20 puntos obtenidos del frame, para ello lo primero que se hace es para cada Joint llamar a la función GetDisplayPosition() para tener las coordenadas del Canvas en donde tiene que ser dibujado. El punto a dibujar será un objeto de la clase Line, para que parezca un punto se le dará la misma longitud que grosor (6). La coordenadas X de origen y destino serán x-3 e x+3 (siendo X la posición del Joint) y la coordenada Y será en origen y destino la misma que la del Joint.

```
private void PaintJoints(SkeletonData skeleton)
{
    foreach (Joint joint in skeleton.Joints)
    {
        var jointPos = _kinectCanvas.GetDisplayPosition(joint);
        var jointLine = new Line
        {
            X1 = jointPos.X - 3
        };
        jointLine.X2 = jointLine.X1 + 6;
        jointLine.Y1 = jointLine.Y2 = jointPos.Y;
        jointLine.Stroke = kinecCanvas.JointColors[joint.ID];
        jointLine.StrokeThickness = 6;
        canvas1.Children.Add(jointLine);
    }
}
```

Figura 4.15 Método PaintJoints

El color del Joint será el que se haya marcado en el diccionario para ese Joint. Una vez creada la línea se añadirá al Canvas.

Una vez dibujados los Joints será necesario añadir los "huesos" del esqueleto, esto lo hace el método PaintBones(), este método tiene como variables de entrada los 20 Joints recogidos por el sensor y el color con el que hay que pintar los huesos (obtenido según el Id del esqueleto del diccionario de la clase KinecCanvas).

Se instanciarán cinco objetos Polyline que unirán los siguientes Joints y formarán el esqueleto:

- Centro cadera, columna, cuello y cabeza
- Cuello, hombro izquierdo, codo izquierdo, muñeca izquierda y mano izquierda
- Cuello, hombro derecho, codo derecho, muñeca derecha y mano derecha
- Centro cadera, cadera izquierda, rodilla izquierda, tobillo izquierdo y pie izquierdo
- Centro cadera, cadera derecha, rodilla derecha, tobillo derecho y pie derecho

4.4.2 Reconocimiento de Posturas

Se quiere conseguir que el programa detecte cuando el jugador se pone una mano en la cabeza. Como el reconocimiento de posturas no está implementado en el SDK de Kinect es necesario implementarlo a partir de los Joints del esqueleto. Lo primero que se hará será añadir un elemento Label al XAML para indicar la postura detectada en cada momento.

La manera de detectar una postura u otra se hace usando la posición de estos y mirando si están en línea, etc. Los métodos que detectan las diferentes posturas están en una clase independiente llamada PostureDetector con el fin de facilitar su posterior reutilización.

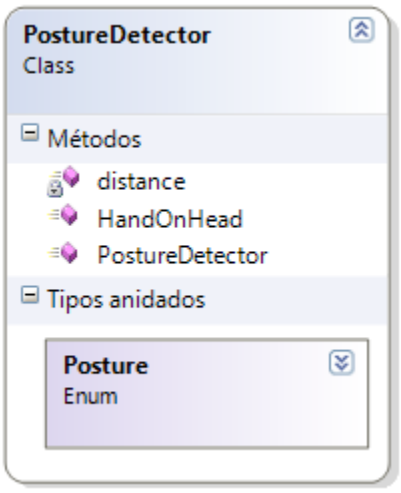


Figura 4.16 Clase PostureDetector

En la clase se pueden destacar los siguientes métodos:

- distance(), se le pasan dos float y calcula la distancia entre ellos, los parámetros de entrada pueden ser positivos o negativos (recordar eje de coordenadas de los Skeleton).
- HandOnHead(), esta clase recibe como variable de entrada dos Joint (el correspondiente a la cabeza y una de las dos manos) y devuelve un booleano, que es verdadero si la mano se encuentra encima de la cabeza. Para determinar si esto se da, lo que se hace es calcular la suma de las distancias (X, Y y Z) de manera que si la suma de las distancias es menor que un valor se considera que los Joints mano y cabeza están en la misma posición y devuelve el valor verdadero.

En la clase principal lo que hará es llamar a la función HandOnHead(), después de haber instanciado la clase PostureDetector en el método MainWindow_Loaded(), y si este método devuelve verdadero escribirá en el Label el nombre de la postura, o "none" si no se ha detectado ninguna.

4.4.3 Reconocimiento de Gestos

Al igual que el detector de posturas la detección de gestos no viene implementada en el SDK de Microsoft por lo que es necesario desarrollarlo. En este caso lo que se quiere conseguir que la aplicación detecte el deslizamiento de la mano a la derecha (sin cambiar de altura) y que muestre el número de veces que sea ha detectado este gesto.

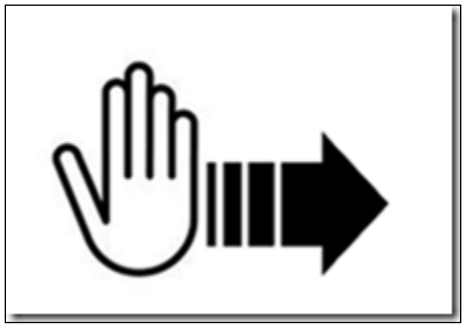


Figura 4.17 Movimiento de deslizamiento

La mecánica de detección de gestos consiste en observar la evolución de uno o varios Joint (en este caso el Joint correspondiente a la mono derecha) en el tiempo. Para ello se irá guardando en un array los datos referentes a su posición. Cada vez que se añada un nuevo dato se comprobará con los datos del array si se ha producido el gesto, para eso se comprobará la evolución del Joint en el espacio y también el tiempo en el que se ha producido, es decir se tiene que dar un desplazamiento satisfactorio en un tiempo determinado.

Para el desarrollo de esta funcionalidad lo primero que se hará será añadir al archivo XAML un elemento Label donde se mostrarán el gesto detectado junto con el número de desplazamientos detectados.

4.4.3.1 Clase GestureDetector

Al igual que con la detección de posturas se implementará una nueva clase llamada GestureDetector donde se encontraran los métodos, propiedades y estructuras necesarias para la detección de gestos y de esta manera facilitar su posterior reutilización en otras aplicaciones.

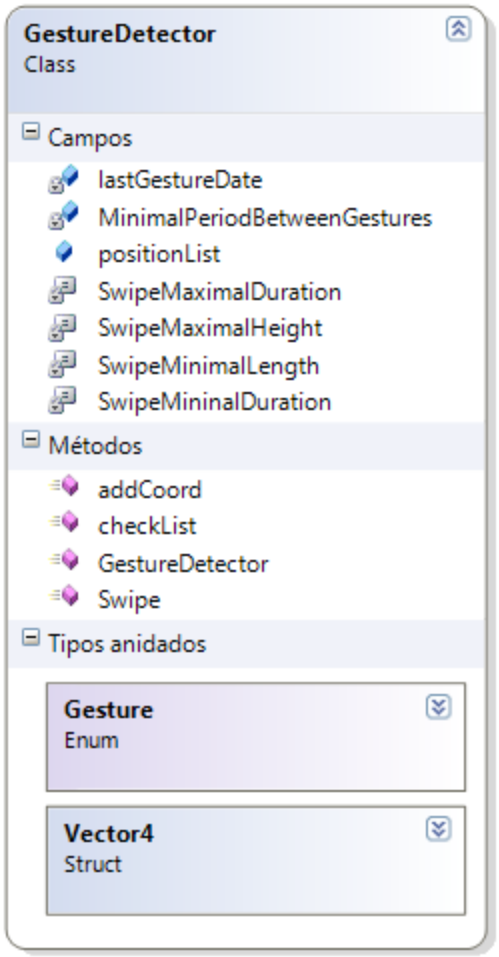


Figura 4.18 Clase GestureDetector

Como se ha comentado anteriormente los datos de la posición de la mano derecha serán guardados en un array, estos datos estarán guardados dentro de una estructura llamada Vector4 que contiene los siguientes datos:

- X, guarda la posición en el X del Joint
- Y, guarda la posición en el Y del Joint
- Z, guarda la posición en el Z del Joint
- date, guarda la fecha y hora en la que se obtuvieron los datos de la posición del Joint

El atributo Gestures no es más que una enumeración con los gestos que se pueden detectar, no tiene ninguna función importante.

El array donde se van guardando los datos es el llamado positionList y es instanciado en el constructor, el resto de atributos de la clase son:

- MinimalPeriodBetweenGestures, como su nombre indica guarda el tiempo que tiene que pasar después de reconocer un gesto para poder aceptar que se ha producido otro.
- lastGestureDate, guarda la fecha y hora de cuando se reconoció el último gesto

El resto de atributos son constantes que determinan las características que debe tener el gesto a reconocer, en este caso el de desplazamiento (o como es llamado en la clase, swipe).

- SwipeMaximalDuration, determina la duración máxima que puede durar el gesto
- SwipeMinimalDuration, determina la duración mínima que puede durar el gesto
- SwipeMaximalHeight, determina la variación máxima en el eje Y medido en metros que puede variar la posición entre la posición de un Joint y el siguiente detectado.
- SwipeMinimalLength, es el desplazamiento a lo largo del eje X medido en metros de un Joint mínimo para considerar que se ha producido un desplazamiento.

Como se puede ver las medidas de longitud se trabaja en metros, esto es debido a que se trabaja con las coordenadas de los Joints y estas vienen en metros.

Los métodos de la clase son los siguientes:

- GestureDetector(), es el constructor de la clase, en este método se instancia el array positionList que es quien guardará la información sobre la evolución de la posición del Joint a lo largo del tiempo.
- addCoor(), recibe como parámetro de entrada el Joint, y guarda los datos correspondientes a su posición, la fecha y hora de ese momento dentro del array positionList en la forma de una estructura Vector4.
- Swipe(), devuelve un booleano en función de si según los datos guardados en la positionList es capaz de detectar que se ha producido un gesto o no (ver funcionamiento en figura 3.19).

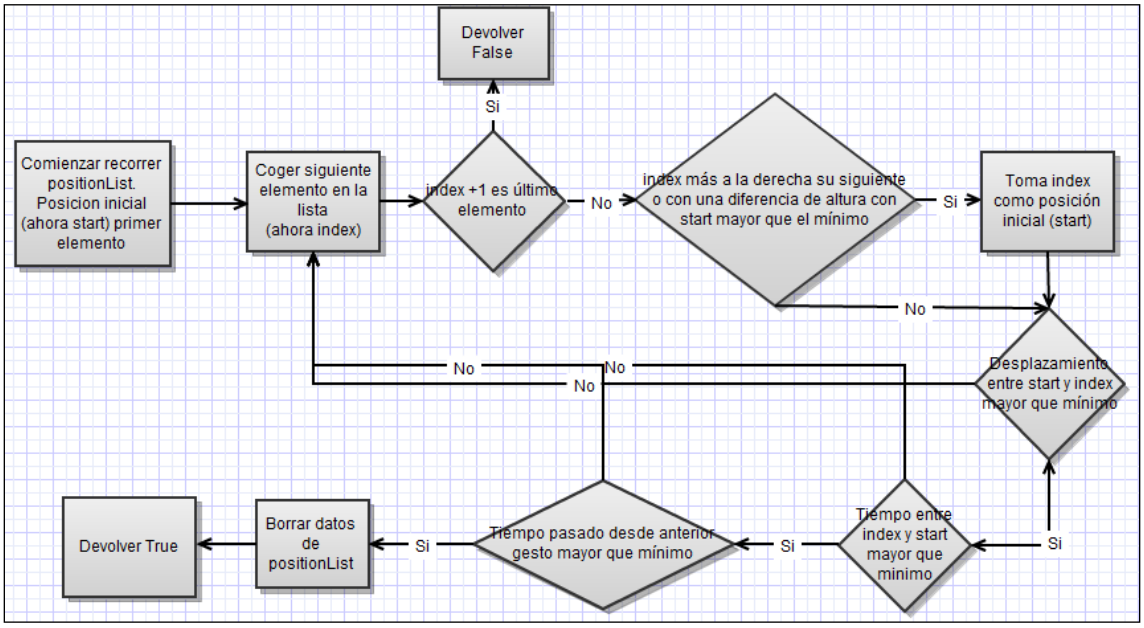


Figura 4.19 Diagrama Flujo método Swipe()

- checkList(), con el objetivo de evitar que el array positionList sea muy grande esta función será llamada después de comprobar si se ha reconocido algún gesto. Comprobará que el array no contenga más de veinte elementos y si es así se borrará el primero, es decir, el más antiguo. Se le llama desde la clase MainWindow.

4.4.3.2 Detección en la clase MainWindow

Como se puede apreciar en la figura 3.20 que representa el código que llama al detector de gestos en la clase MainWindow (evidentemente se había instanciado el objeto GestureDetector ya en el método MainWindow_Loaded()) el funcionamiento es como se ha expuesto ya, primero se añade la posición del actual de Joint al array (mediante el método addCoor()), una vez hecho esto se llama al método Swipe que dice si ha detectado el gesto de deslizamiento, de ser así mostrará en el elemento Label creado anteriormente el número de gestos de deslizamiento detectados. Al final del todo llama al método checkList() con el objetivo de evitar que el array se haga demasiado grande.

```
//deteccion gestos
gestureDetector.addCoord(skeleton.Joints[JointID.HandRight]);
if (gestureDetector.Swipe())
{
    gestureAcceptedList.Add(GestureDetector.Gesture.Swipe);
    label2.Content = "Swipe " + gestureAcceptedList.Count;
}
gestureDetector.checkList();
```

Figura 4.20 Código que llama al detector de gestos

4.4.3.3 TransformSmoothParameters

Por último y con el objetivo de reducir las vibraciones producidas por la actualización de los frames se utiliza la estructura TransformSmoothParameters. Los valores que se han usado son los que se usan por defecto pero habría que modificarlos según las necesidades. El código de instanciación de la estructura (ver figura 3.21) se incluyen dentro del método MainWindow_Loaded.

```
var parameters = new TransformSmoothParameters
{
    Smoothing = 0.75f,
    Correction = 0.0f,
    Prediction = 0.0f,
    JitterRadius = 0.05f,
    MaxDeviationRadius = 0.04f
};
kinect.SkeletonEngine.SmoothParameters = parameters;
```

Figura 4.21 Código de TransformSmoothParameters

Los campos que tiene la estructura son los siguientes:

- Smoothing, establece u obtiene la cantidad de suavizado, la cantidad oscila entre [0,1.0].
- Correction, establece u obtiene la cantidad de corrección, la cantidad oscila entre [0,1.0] siendo por defecto 0.5.
- Prediction, establece u obtiene el número de frames previsto.

- JitterRadius, obtiene o establece el radio de reducción de vibración (en metros), el predeterminado es 0.05m (5cm).
- MaxDeviationRadius, establece u obtiene el radio máximo que las posiciones de filtrado pueden diferir de los datos en bruto. El valor predeterminado es 0.04.

El resultado final que nos ofrece la aplicación es el que muestra la figura 3.22:

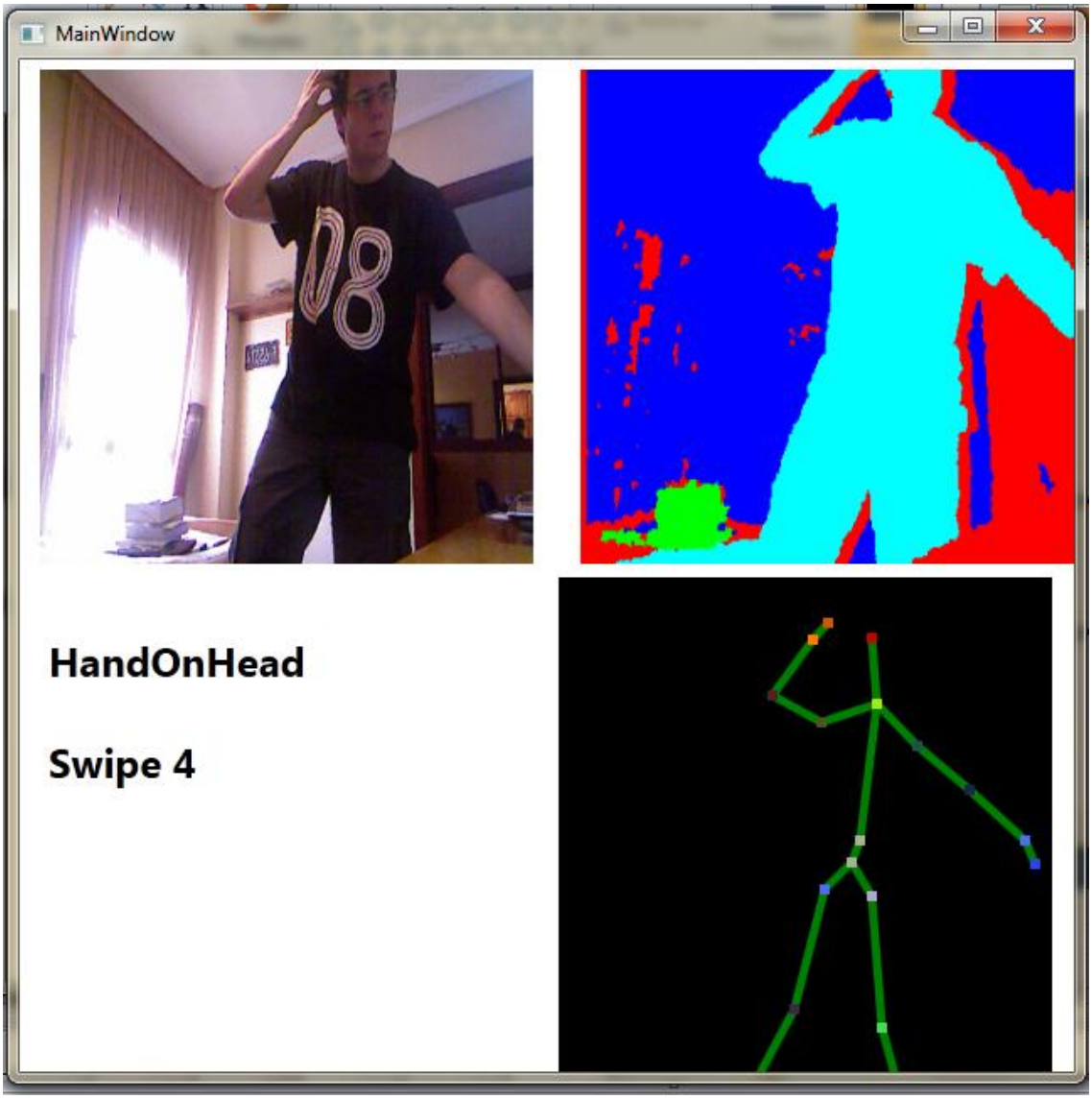


Figura 4.22 Resultado mini-aplicación 2

4.5 Tercera mini-aplicación: Pintar la pantalla

El objetivo de esta pantalla es pintar lo que dibuje el usuario con una mano. El objetivo es que mientras el jugador se esté tocando la cabeza ir dibujando el movimiento de la mano derecha. Para borrar lo dibujado habrá que realizar un movimiento de Push con la mano derecha.

Se trabajará a partir del código creado en las aplicaciones anteriores (aunque habrá que modificarlo).

En el fichero XAML los elementos Canvas para mostrar el esqueleto y el Image para mostrar profundidades no se eliminaran ya que servirán como orientación (para ver si el sensor ha detectado al jugador) pero tendrán un tamaño menor y estarán en el lado izquierdo. El objeto Image que muestra lo detectado por la cámara RGB se dejará en el centro y se le dará un tamaño mayor. Para dar la sensación de que se está pintando sobre la imagen a color al elemento Image se le superpondrá un elemento Canvas de las mismas dimensiones donde se dibujará el dibujo que haga el usuario, no se le dará color de fondo, de esta manera se conseguirá el efecto deseado. También se creará un objeto Elipse que se mostrará cuando el jugador se toque la cabeza, su aparición mostrará que el modo pintar estará activado.

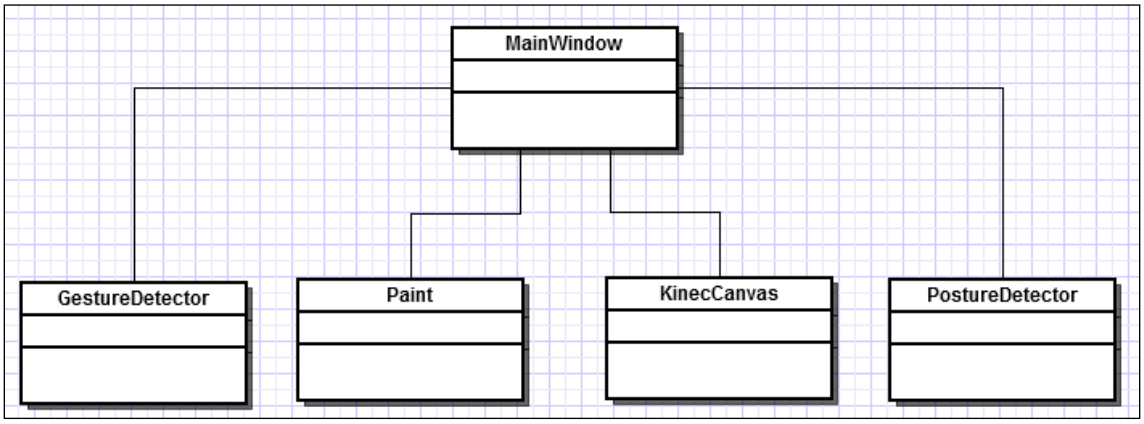


Figura 4.23 Diagrama de clases mini-aplicación 3

Como se puede ver en la figura 3.23 se mantienen las mismas clases que se tenían en las aplicaciones anteriores, se ha añadido la clase Paint que será la encargada de dibujar en el Canvas, las clases KinecCanvas y PostureDetector no son modificadas.

4.5.1 La clase GestureDetector

En esta clase es necesario añadir un nuevo gesto a detectar, el Push (empujar), al hacerlo se limpiará el contenido del Canvas. Este gesto será detectado por la función Push() que devolverá verdadero cuando se detecte el gesto y falso cuando no se detecte.

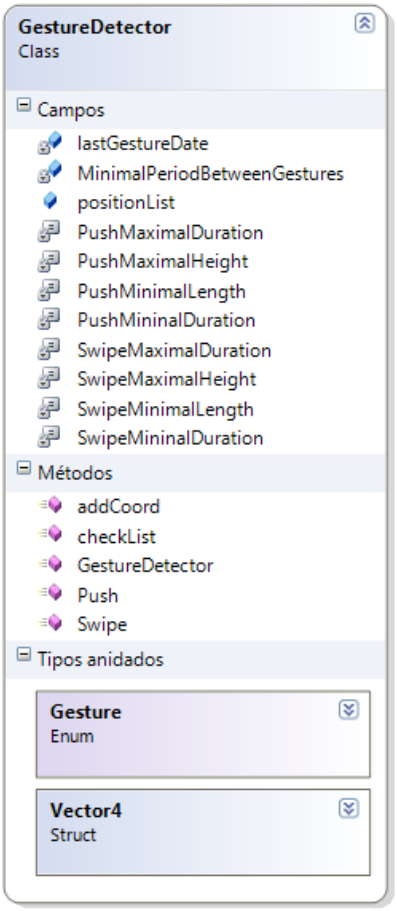


Figura 4.24 Clase GestureDetector

Como se puede ver en la figura 3.24, la clase y su funcionamiento es el mismo que el explicado en la aplicación anterior, simplemente se han añadido la función Push() y las constantes que describen el gesto. Estas constantes son:

- PushMaximalDuration, 1500 milisegundos
- PushMaximalHeight, 0.2 metros
- PushMinimalDuration, 250 milisegundos
- PushMinimalLenght, 0.1 metros

La función Push() funciona de manera similar a la función Swipe() con dos diferencias:

- Ahora se controla que el elemento actual de la lista (o index) no esté más cercano al sensor que el elemento siguiente (antes se comprobaba que no estuviera más a la derecha).
- Como es obvio el desplazamiento mínimo para considerar que se ha producido un gesto de Push se mide en el eje Z en vez de en el eje X, es decir el movimiento hacia el sensor.

4.5.2 Clase Paint

Esta clase es la que se encargará de pintar en el Canvas. El funcionamiento es el siguiente, cuando se añade una nueva coordenada, se comprueba si es la primera que se dibuja, de ser así se dibuja un punto (similar al de los correspondientes a los Joints cuando se dibuja el esqueleto) si no lo es, se dibuja una línea que va desde la coordenada que se ha leído anteriormente (es necesario guardarla) hasta la nueva coordenada.

Como se puede ver en la figura 3.25 se ha creado una estructura llamada Vector2, esta estructura tiene dos elementos y servirá para guardar las coordenadas de los puntos a dibujar.

Los atributos de esta clase son:

- _canvas, es una referencia al objeto Canvas donde se va a dibujar.
- Elipse, es una referencia al objeto elipse que se muestra visible cuando se está tiene el jugador la mano en la cabeza, sirve para indicar que se puede comenzar a dibujar.
- kinect, es una referencia al sensor Kinect
- ultimoPunto, es donde se guardan los datos del último punto dibujado, se guardan dentro de una estructura Vector2. Las coordenadas están ya adaptadas al Canvas (se hace de la misma manera a como se adaptan las coordenadas de los Joints para dibujarlos en el esqueleto).
- esNuevo, variable que dice si el punto a dibujar es el primero o no (para dibujar un punto o una línea), si es true, significa que es el primer punto.

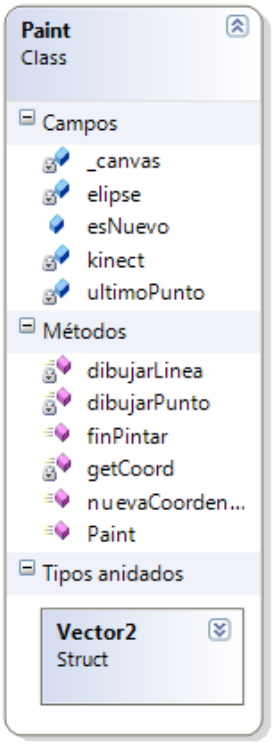


Figura 4.25 Clase Paint

Los métodos de esta clase son:

- Paint(), es el constructor de la clase, recibe como parámetros de entrada las referencias al sensor Kinect y a los elementos Elipse y Canvas.
- nuevaCoordenada(), recibe como parámetro de entrada el Joint correspondiente a la mano derecha, es el encargado de llamar a otros métodos de la clase para pintar en el Canvas y de poner visible el elemento elipse.
- getCoord(), es invocado por la función nuevaCoordenada(), recibe como parámetro un Joint y devuelve una estructura Vector2 que contiene la información sobre la posición de este Joint adaptada al Canvas donde se va a dibujar.
- dibujaPunto(), invocado por el método nuevaCoordenada() cuando es el primer punto que hay que dibujar, recibe como parámetro de entrada una estructura con la información sobre la posición del punto. Dibuja el punto de la misma manera que se dibujan las articulaciones del esqueleto (con la clase Line).
- dibujaLinea(), es invocado por el método nuevaCoordenada() cuando el punto a dibujar no es el primer punto a dibujar, recibe como parámetro de entrada una estructura Vector2 con la información sobre la posición de la coordenada detectada. Su función es dibujar una línea (clase Line) que irá desde este punto hasta el anterior punto detectado (guarda su información el atributo ultimoPunto).
- finPintar(), es un método que es llamado cuando el jugador deja de tocarse la cabeza (momento en que se desactiva el modo pintar), es llamado desde la clase MainWindow y lo que hace es ocultar la elipse y poner la variable esNuevo a true para indicar que el próximo punto que haya

que leer (una ver el jugador vuelva a ponerse la mano en la cabeza) será considerado como el primer punto a dibujar.

A excepción de los métodos finPintar() y el constructor el resto de los métodos son llamados por el método nuevaCoordenada() por lo que es sencillo explicar el funcionamiento conjunto de la clase con un diagrama de secuencia de este método (figura 3.26).

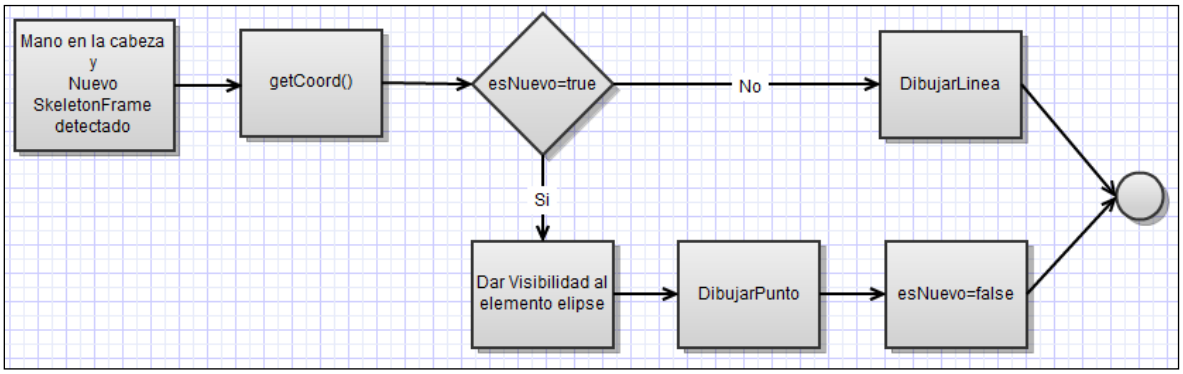


Figura 4.26 Diagrama de secuencia de nuevaCoordenada()

4.5.2 La clase MainWindow

Respecto de las otras aplicaciones hay que hacer muy pocos cambios en esta clase, solamente la instanciación de la clase Paint que se hará en el MainWindow_Loaded y el código correspondiente al manejo de esta clase, este código ira dentro del método kinect_SkeletonFrameReady().

```
if (postures.isHandOnHead(skeleton.Joints[JointID.HandLeft], skeleton.Joints[JointID.Head]))
{
    pizarra.nuevaCoordenada(skeleton.Joints[JointID.HandRight]);
}
else
{
    pizarra.finPintar();
    //introducimos detector de gestos
    gestures.addCoord(skeleton.Joints[JointID.HandRight]);
    if (gestures.Push())
    {
        canvasPaint.Children.Clear();
    }
    gestures.checkList();
}
```

Figura 4.27 Código de gestión de la clase Paint

Como se puede ver en la figura 3.27 lo que se hace es una vez se ha detectado un nuevo SkeletonFrame si el usuario tiene la mano izquierda sobre su cabeza se usa el método nuevaCoordenada() de la clase Paint (instanciado en pizarra) y se pone en marcha el proceso explicado en la figura 3.26. Si por el contrario el usuario no tiene la mano en la cabeza se llama al método finPintar() para asegurarse de que ni se va a ver la elipse y que

el próximo punto a pintar será el primero, además se añadirá la posición de la mano al detector de gestos y se comprobará si se ha producido el gesto Push, de ser así se limpiará el Canvas, es decir, se borrará lo pintado.

El resultado final de la aplicación es el siguiente:

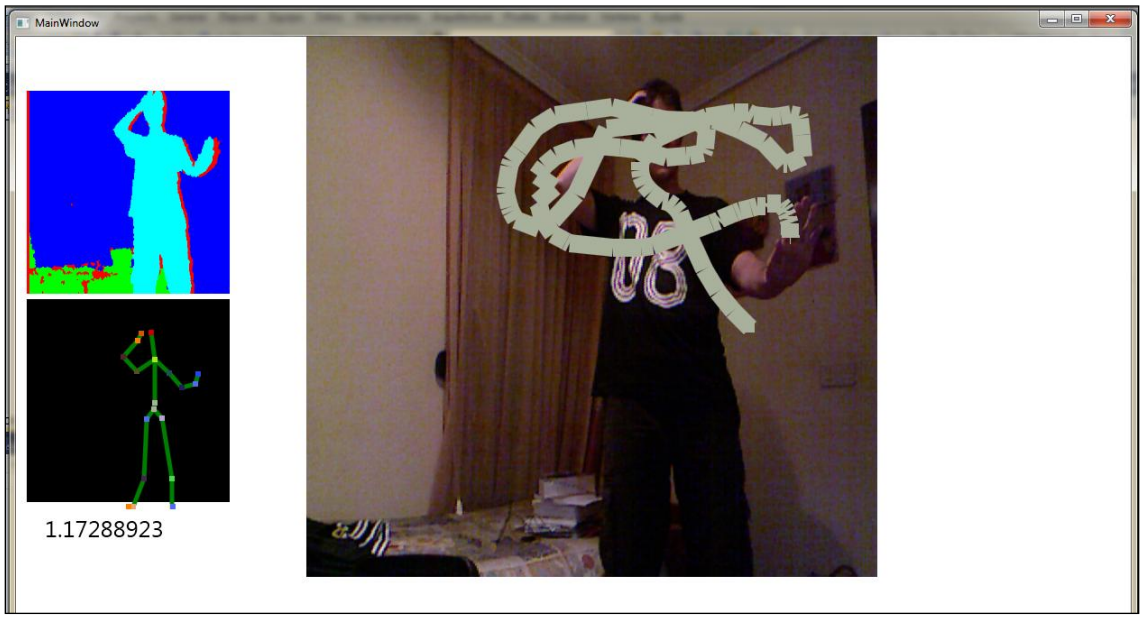


Figura 4.28 Resultado mini-aplicación 3

4.6 Cuarta mini-aplicación reproductor de sonidos

El objetivo de esta aplicación es integrar Kinect con la reproducción de audio, el objetivo es dibujar ciertos elementos encima de la imagen captada con el sensor (usando la misma técnica de poner un Canvas encima del entorno Image que se usa en el ejemplo anterior).

Dentro de la imagen se dibujaran los siguientes elementos (ver figura 3.29):

- Dos elementos Line a mano derecha que al pasar la mano por encima sonará un sonido similar a una guitarra.
- Una elipse debajo de los dos elementos Line que al tener la mano sobre él y hacer un gesto Push sonará como un bombo.
- Un elemento Line (posición vertical) a mano izquierda, al poner la mano encima y según a la altura a la que se encuentre sonará un sonido con más o menos volumen.

La imagen con la disposición final es la siguiente:

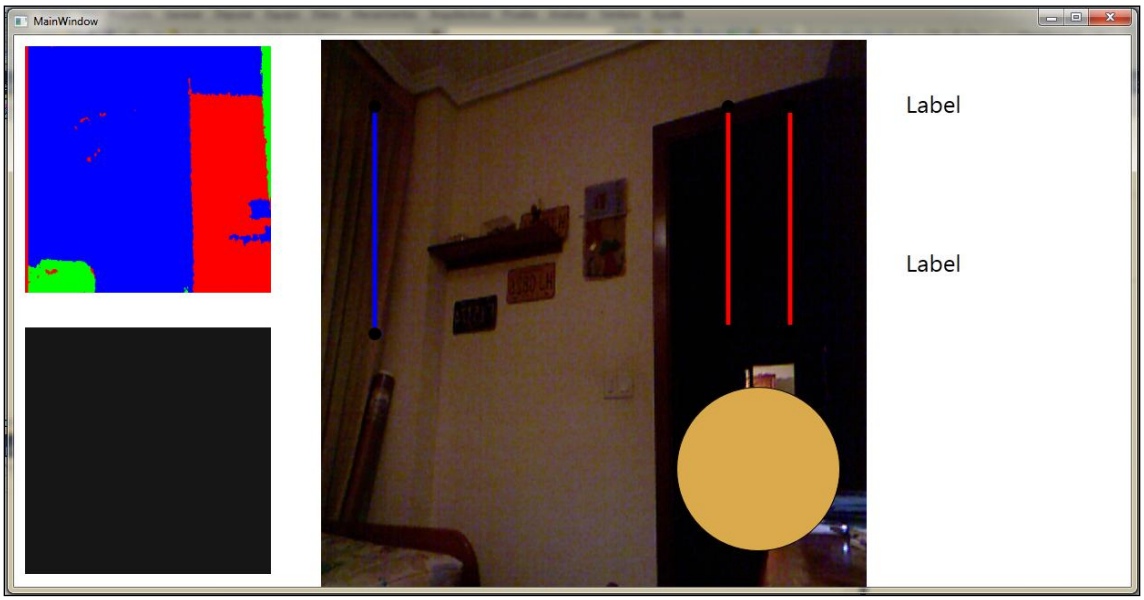


Figura 4.29 Imagen final mini-aplicación 4

A la izquierda de la pantalla aparecen dos elementos que mostraran la imagen de profundidad y el esqueleto del jugador, al igual que la aplicación anterior servirá de orientación para ver si el jugador ha sido detectado correctamente.

La estructura de clases de la aplicación será la recogida en la figura 3.30.

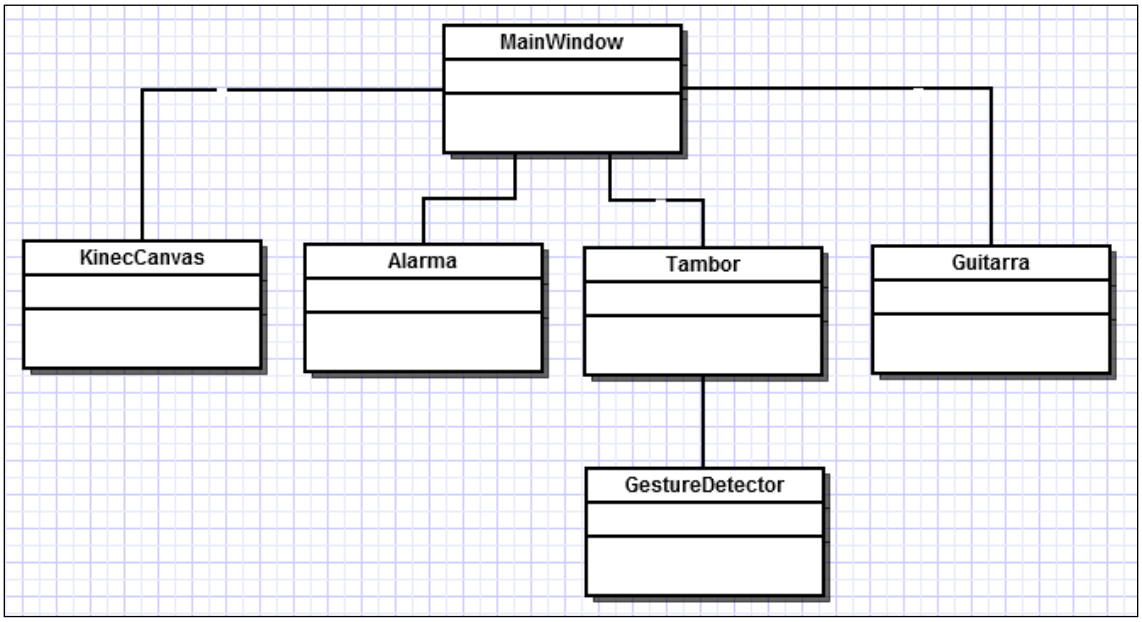


Figura 4.30 Diagrama de Clases mini-aplicación 4

Como se puede ver en el diagrama de clases se reutilizan las clases GestureDetector y KinecCanvas usadas en las aplicaciones anteriores, el resto de nuevas clases llevarán a cabo la reproducción de sonido de

cada elemento de los que se ha situado en el Canvas encima de la Image donde se verán las imágenes RGB captadas por el sensor.

4.6.1 Reproducción de sonidos en WPF

La versión 3.0 de .NET Framework en el año 2007 (incluida dentro de Windows Presentation Foundation), trajo dos controles que simplificaban el audio para aplicaciones Windows. Ambos dos son una interfaz que se comunica con funcionalidad existente en Win32 y Windows Media Player. Las dos principales clases son:

- SoundPlayer
- Media Player

SoundPlayer se encuentra en la biblioteca System.Media que forma parte de .NET framework desde su versión 2.0. Esta clase sólo soporta archivos .WAV y no puede reproducir múltiples sonidos al mismo tiempo, dándose una interrupción en el audio cuando se quieren reproducir varios sonidos, ni tampoco soporta el cambio de volumen. Sin embargo es la forma más sencilla de reproducción, basta con instanciar un objeto de tipo SoundPlayer con el nombre del archivo a reproducir e invocar el método Play.

"SoundPlayer player= new SoundPlayer("canción.wav");

player. Play();"

Adicionalmente si se quieren reproducir sonidos propios del sistema ya que el espacio de nombres System.Media tiene una clase que permite acceder a estos sonidos. Además XAML permite la reproducción de sonidos vinculados directamente a acciones que puedan suceder en respuesta de eventos de controles de interface gráfica. La manera más sencilla es utilizarlos a través de un trigger denominado "SoundPlayerAction" que recibe como parámetro la fuente del archivo que se desea reproducir

MediaPlayer es la clase más utilizada debido a que tiene más funcionalidades. Está basado en las bibliotecas de Windows Media Player por lo que permite soportar los formatos que puede leer dicho reproductor. Da la posibilidad de controlar el volumen, silenciar el sonido, mover un balance de sonidos a un parlante basado en un valor doublé y además ofrece métodos capaces de pausar, reanudar/comenzar y detener la reproducción. Adicionalmente si el formato de audio lo soporta permite aumentar o disminuir la velocidad de lo reproducido y operaciones de posicionamiento y búsqueda sobre el sonido en reproducción. La reproducción se puede hacer de manera asíncrona y una sola clase puede reproducir varios archivos.

Si se desea utilizar la clase MediaPlayer en el XAML, se tiene la clase MediaElement. Tiene eventos que facilitan el manejo de errores que se puede presentar porque falta el archivo o por problemas al reproducir.

4.6.2 Clase Tambor

Esta clase gestionará la reproducción del sonido del tambor, cuando se tenga la mano sobre el tambor y se haga Push se reproducirá el sonido de un tambor.

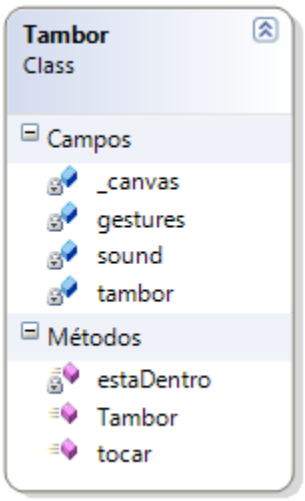


Figura 4.31 Clase Tambor

Como se puede ver en la figura 3.31 la clase tambor tiene los siguientes atributos:

- _canvas, es una referencia al Canvas donde están dibujados los elementos.
- gestures, es una instancia de la clase GestureDetector, esta clase se usa para detectar si se ha producido gesto Push.
- sound, es una instancia de la clase MediaElement que reproducirá el sonido.
- tambor, es una referencia a la elipse que representa el tambor en el Canvas.

Análogamente los métodos son:

- Tambor(), es el constructor de clase recibe como parámetros de entrada la elipse que representa al tambor, el Canvas donde está dibujada y el MediaElement que reproducirá el sonido.
- estaDentro(), comprueba si el Joint que representa a la mano está dentro del dibujo de la elipse.
- tocar(), método al que hace llamada la clase principal para ver si se tiene que reproducir el sonido, primeramente comprueba si el Joint está dentro de la elipse y si se ha producido un movimiento Push reproduce el sonido que se debe reproducir.

```
public void tocar(Point mano, Joint joint)
{
    gestures.addCoord(joint);
    if (estaDentro(mano))
    {
        if (gestures.Push())
        {
            sound.Position = TimeSpan.Zero;
            sound.Play();
        }
    }
    gestures.checkList();
}
```

Figura 4.32 método Tambor.tocar()

4.6.3 Clase Guitarra

Gestionará el elemento guitarra. Esa clase podrá representar a una o más líneas que representarán las cuerdas de una guitarra. El sonido sonará al entrar en contacto el Joint con la Linea que representa a la cuerda.

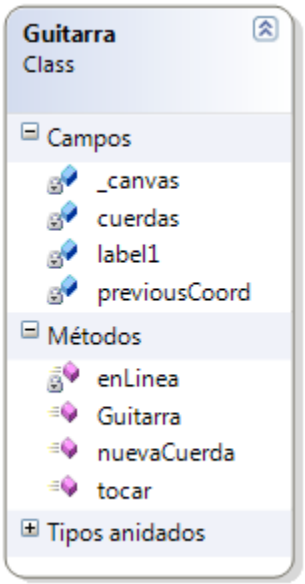


Figura 4.33 Clase Guitarra

Como se puede ver la clase tiene los siguientes atributos:

- _canvas, es una referencia al Canvas donde están dibujadas las líneas que representan las cuerdas.
- cuerdas, es una lista de estructuras Linea.
    - Linea, es una estructura de datos propia que representa los datos de una cuerda, contiene los siguientes elementos:
        - cuerda, elemento de tipo Line, es la representación gráfica.
        - or, elemento Point donde se sitúa el origen de la cuerda.
        - des, elemento Point donde se sitúa el final de la cuerda.

    - sound, elemento MediaElement que reproducirá el sonido.
- previousCoord, debido a que el sonido se reproduce al tocar las cuerdas para evitar que si se deja la mano encima de la cuerda esté reproduciendo indefinidamente, se guarda la anterior coordenada.

Los métodos implementados por la clase son los siguientes:

- Guitarra(), es el constructor de la clase, recibe como parámetro el Canvas donde se dibujan las líneas.
- nuevaCuerda(), se creará una nueva cuerda, recibe como parámetros de entrada los Point que guardan los datos de origen y destino de la cuerda y una referencia al MediaElement que reproducirá el sonido.
- enLinea(), dados un Point y un elemento Linea, el método devuelve un booleano diciendo si ese Point está en la línea o no.
- tocar(), método invocado desde la clase principal para ver si se debe reproducir o no el sonido. Su funcionamiento consiste en ver si el Joint está en contacto con la línea y reproducir el sonido de ser así.

4.6.4 La clase Alarma

Esta clase gestionará el elemento alarma, se trata de un elemento Line en el que según a la altura a la que se coloque la mano sonará a más o menos volumen. En esta clase también se utilizará el elemento Linea explicado en el punto anterior.

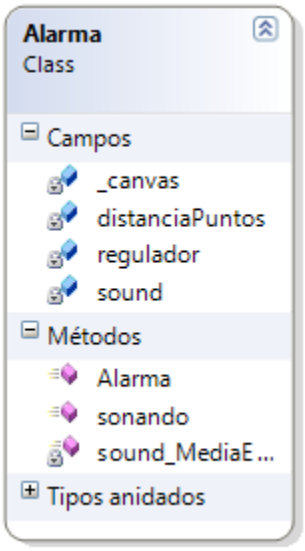


Figura 4.34 Clase Alarma

Los atributos de la clase son los siguientes:

- _canvas, es una referencia al Canvas donde se dibuja la línea.
- distanciaPuntos, es el tamaño de la línea.
- sound, es el MediaElement que reproduce los sonidos.

- regulador, es el elemento Linea que guarda los datos del dibujo de la Linea.

Los métodos de la clase son los siguientes:

- Alarma(), es el constructor de la clase, recibe como parámetros de entrada la referencia al Canvas, las coordenadas de los puntos origen y destino de la línea (guardados en estructuras tipo Point) y el elemento MediaElement que reproducirá el sonido.
- Sonando(), es el método que se llama desde la clase principal, este método comprueba si la mano está encima de la línea, de ser así comienza la reproducción o continua con ella poniendo el volumen en función de la altura de la mano, si la mano no está en la línea, para la reproducción.
- sound_mediaEnded, método que se llama (evento) cuando se acaba la reproducción del sonido, vuelve a reproducir el sonido desde el principio.

4.6.5 La clase principal

La clase MainWindow no cambia prácticamente, sólo se cambia el método Kinect_SkeletonFrameReady():

```
void kinect_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
    int skeletonId = 0;
    canvasSkeleton.Children.Clear();
    foreach (SkeletonData skeleton in e.SkeletonFrame.Skeletons)
    {
        skeleton.Quality = SkeletonQuality.ClippedBottom;
        if (skeleton.TrackingState == SkeletonTrackingState.Tracked)
        {
            //kinec solo puede detectar dos esqueletos
            PaintBones(skeleton, KinecCanvas.SkeletonColor[skeletonId]);
            PaintJoints(skeleton);
            skeletonId++;

            guitar.tocar(changeEscale(skeleton.Joints[JointID.HandRight]));
            tambor.tocar(changeEscale(skeleton.Joints[JointID.HandRight]),skeleton.Joints[JointID.HandRight]);
            alarma.sonando(changeEscale(skeleton.Joints[JointID.HandLeft]));
        }
    }
}
```

Figura 4.35 Clase kinect_SkeletonFrameReady

Como se puede ver en la figura el funcionamiento es similar que en las aplicaciones anteriores sustituyendo el tratamiento de gestos y posturas que se hace después de dibujar el esqueleto por la llamada a los métodos tocar y sonando de las clases explicadas en los anteriores puntos, estas clases se instancian en el método MainWindow.

Los MediaElement que se usan para reproducir sonido serán creados en el archivo XAML y ahí se indicaran el archivo de sonido que deben reproducir.

Capítulo 5 Kinect aplicado a la educación infantil

El objetivo de esta parte es el desarrollo de una aplicación educativa en la que se pueda hacer uso de las ventajas de interacción que ofrece Microsoft Kinect.

Para el desarrollo de esta aplicación se ha decidido dejar de usar WPF y usar el Framework de Microsoft para el desarrollo de juegos, XNA.

La razón por lo que se ha decidido usar XNA en vez de WPF es que el primero al ser un framework creado específicamente para la creación de videojuegos facilita la creación de estos en comparación con WPF (creado para hacer interfaces gráficas). Estas mejoras son, entre otras, una mayor sencillez para trabajar con imágenes y las posibles colisiones con ellas, la facilidad para reproducir sonidos y que ya viene implementado el bucle de juego, lo que facilita mucho la labor del programador. Además con XNA hace posible el desarrollo de juegos compatibles con la consola Microsoft Xbox 360, que permitiría en un futuro (cuando Microsoft publique un SDK de Kinect compatible con ésta) hacer la aplicación compatible con la consola.

5.1 Microsoft XNA

Microsoft XNA (XNA's Not Acronymed o Xbox New Architecture. Es un conjunto de herramientas con un entorno de ejecución suministrado por Microsoft con el objetivo de facilitar el desarrollo de juegos. En un intento de liberar a los creadores de la creación de código repetitivo, y conseguir unir diferentes aspectos de la producción de un juego en un único sistema.

En agosto de 2006 Microsoft liberó el XNA Framework y el Game Studio (IDE, ahora también se encuentra integrado en Visual Studio).

El Framework XNA está basado en la implementación del .NET Compact Framework 2.0 para Xbox 360 y el .NET Framework 2.0 para Windows. Incluye un extenso conjunto de librerías de clases, específicas del desarrollo de videojuegos, para promover la máxima reutilización de código entre diferentes plataformas.

Uno de los objetivos de XNA es facilitar el desarrollo de juegos para diferentes plataformas (plataformas Microsoft). Un juego compilado en una plataforma no sirve para otra ya que se ha utilizado distinto framework .NET (Compact en el caso de Xbox). Sin embargo al ser compatible con los dos framework es posible que el código generado (antes de compilar) para Windows sea compatible para la Xbox, aunque evidentemente hay funcionalidades que no son compatibles para todas las plataformas

Aunque teóricamente XNA acepta todos los lenguajes de programación de la plataforma .NET, en las versiones de express de Game Studio sólo soporta el C#, en el resto de las versiones sólo están soportados C# y desde 2011 Visual Basic .Net.

Figura 5.1 Modelo de capas de XNA

La licencia del Framework de XNA prohíbe la distribución de juegos que se conecten a Xbox Live y/o Windows Live sin haber un acuerdo previo entre el desarrollador y Microsoft. Esto quiere decir que XNA Game Studio puede ser utilizado para el desarrollo de juegos comerciales y otros programas para la plataforma Windows, aunque el Soporte Xbox/Windows live no puede usarse.

Un proyecto XNA está formado por dos proyectos diferenciados.

- Un primer proyecto donde están las clases que contienen la lógica de la aplicación.
- Un segundo proyecto que contiene todos los contenidos tipo imágenes, texturas, sonidos, etc. (tiene el nombre del primer Proyecto añadiéndole Content).

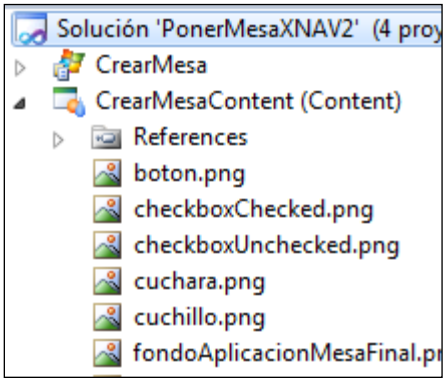


Figura 5.2 Estructura Proyecto XNA

5.1.1 Funcionamiento de XNA

Durante la ejecución se produce una secuencia que se repite constantemente hasta que se produce el evento de fin del juego.

La idea principal del juego es que hay una secuencia de métodos que se ejecutan de manera repetitiva (loop). Para un videojuego esto es importante ya que aunque no haya interacción el juego debe seguir ejecutándose.

Al crear un nuevo Proyecto XNA, se generan dos ficheros de código fuente, el primero es Program.cs y el segundo es Game1.cs. El fichero Program.cs es donde se encuentra el método estático Main, al ejecutarse el videojuego se ejecuta el método que instancia la clase Game1 y llama a su método Run() que es el que inicia la ejecución del juego (el "loop").

El archivo Game1.cs hereda de la clase Game (clase perteneciente al espacio de nombres *Microsoft.Xna.Framework,* que contiene las clases más importantes de XNA. Esta clase Game1 contiene los diferentes métodos que son llamados a lo largo del "loop".

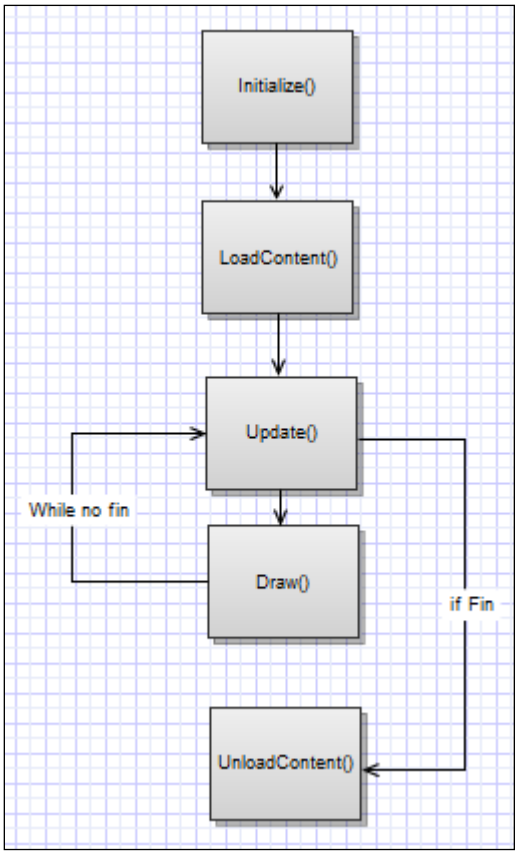


Figura 5.3 Diagrama de funcionamiento XNA

Podemos dividir estos métodos en tres grupos:

- Métodos de preparación del entorno.
  - El método Initialize() se ejecuta al llamar al método Run de la clase Game, aquí se inicia todo lo necesario antes de iniciar el juego (por ejemplo nosotros cargaremos Kinect aquí.
  - El método LoadContent() es el segundo método llamado en segundo lugar, aquí se cargarán los recursos (sonido, imágenes).

    Cuando un proyecto se compila se comprueba que este en un formato y en un directorio conocidos, para garantizar todos estos requisitos es necesario añadir desde Visual Studio estos contenidos y no desde el explorador del sistema operativo. Una vez se encuentra el contenido este es procesado y compilado para que puedan ser utilizados en tiempo de ejecución por el gestor de contenidos de XNA.

- Métodos del Loop, estos métodos son llamados por defecto con una frecuencia de 60 veces por segundo, aunque es una frecuencia regulable.
  - Update(),Es el encargado de leer los datos de entrada y realizar los cálculos, es también el que determina si hay que salir del loop. Por defecto los datos de entrada no se almacenan en un buffer, esto es, podemos saber por ejemplo la posición del ratón o la tecla pulsada en un instante, pero no los datos del instante anterior.
  - Draw(), Es el encargado de renderizar el contenido, los objeto se van pintando en orden en el que aparecen las instrucciones, uno encima de otro, es decir, el último objeto en dibujarse será el que esté encima de todos.

- Método de cierre, son invocados al salir del Loop, cuando se va a cerrar el programa.
  - UnloadContent(), se encarga de las rutinas de finalización.

Kinect no tiene soporte oficial para XNA, sin embargo sí que es perfectamente funcional integrar XNA con Kinect. Nuestra forma de programar con Kinect es usando eventos (como se ha explicado en capítulos anteriores), con esta forma de programar hay que tener cuidado ya que al tener ambas tecnologías ejecuciones periódicas que no podemos coordinar (30fps Kinect y 60 ejecuciones por segundo XNA) es posible que se tengan conflictos entre el Update y los eventos de Kinect al intentar actualizar a la vez las mismas variables, por ejemplo que se esté actualizando el esqueleto mientras se está en el método Update.

5.2 La aplicación, objetivos

El objetivo de la aplicación es realizar un prototipo de herramienta que ayude a los niños, especialmente a aquellos con trastorno generalizado de desarrollo, el aprendizaje de tareas cotidianas, al estilo de los desarrollados en la página web http://www.autismgames.com.au/, en el caso de este proyecto el niño tendrá que poner la mesa de la manera que se ha configurado previamente por un adulto usando el programa generador de misiones también desarrollado.

Para el control del juego se utilizarán las posibilidades de interacción que ofrece Microsoft Kinect, de esta manera se intentará conseguir un juego más usable, con una curva de aprendizaje corta, en definitiva que sea más atractivo para los usuarios,

Las necesidades hardware de la aplicación son las mismas que las que teníamos en los ejemplos de puntos anteriores, es decir, un PC con Windows 7 instalado (no sirve una máquina virtual) y el dispositivo Microsoft Kinect, opcionalmente se puede usar también una pantalla externa, como un proyector o una televisión para así poder disfrutar de una imagen mayor.

Estos elementos se pueden encontrar en cualquier casa o colegio de manera que este sistema permitirá trabajar a los niños tanto en casa como en el colegio.

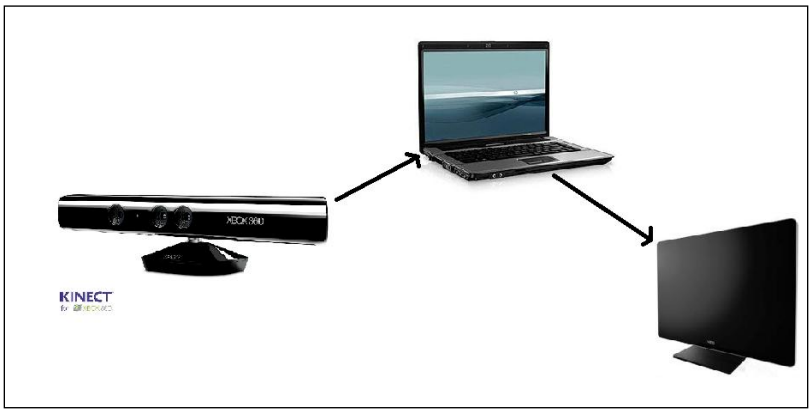


Figura 5.4 Sistema físico

La aplicación que se desarrollará tiene dos partes:

- La aplicación de Poner la Mesa, esta aplicación es el juego, aquí el usuario deberá poner el conjunto de elementos según una disposición que viene descrita en un archivo XML.

- El generador de misiones, aplicación gráfica donde se seleccionará la disposición que se quiere dar a la mesa, este programa generará automáticamente un archivo XML en función de la disposición de los objetos elegida.

5.3 La aplicación PonerMesa, desarrollo

El desarrollo de la aplicación se puede dividir en dos partes, por un lado está el desarrollo de la parte del juego y por otro lado está el desarrollo de la interacción con el sistema.

Como se puede ver en la siguiente figura, las clases que ese encargan de la interacción con Kinect están separadas del resto del sistema, esto se debe a la necesidad de intentar aislar lo máximo posible el resto del sistema de la interacción con Kinect. Esto es debido principalmente a dos razones:

- Al querer evaluar diferentes formas de interacción se tendrán diferentes versiones de estas clases o incluso clases distintas, se desea (para facilitar nuestro trabajo y futuras modificaciones) que estos cambios no afecten al resto de la aplicación, aunque para cambiar de una forma de interacción a otra habrá que modificar el archivo Game1.cs.

- Debido a que el SDK utilizado es una beta por lo que es de esperar que aparezcan mejoras y nuevas funcionalidades, al tener la parte de interacción con el Kinect aislada se podrá actualizar las clases de manera más sencilla. Incluso se podrá cambiar de SDK (para usar OpenNI por ejemplo) sin que el resto de la aplicación se vea afectada.

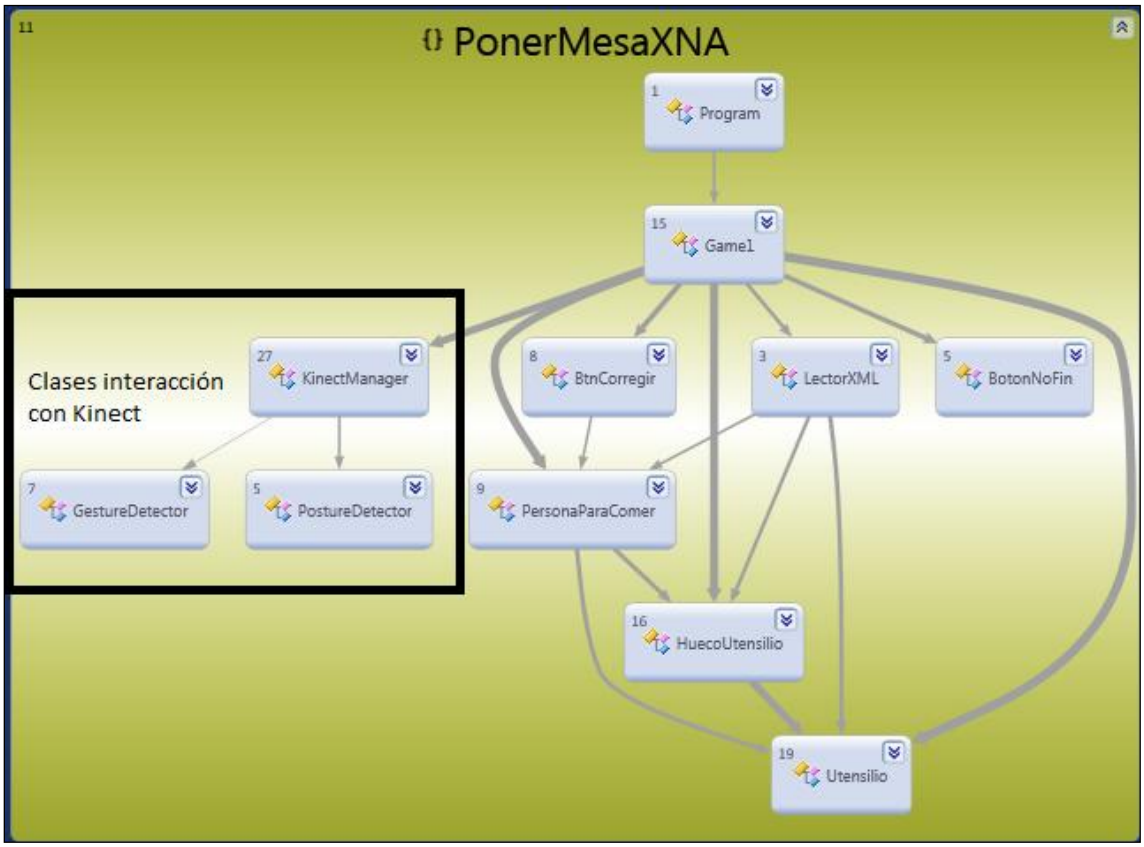


Figura 5.5 Diagrama de Clases PonerMesaXNA

5.3.1 Desarrollo del juego

El objetivo es realizar una aplicación que simule la acción de poner una mesa, para ello se tienen una serie de elementos (tenedor, cuchara, platos…) que se deberán colocar en una mesa. La disposición correcta de los diferentes utensilios vendrá descrita en un fichero XML que se creará a partir de una interfaz gráfica en el que los usuarios podrán seleccionar la disposición ideal de los elementos.

El juego final es el que aparece en la figura siguiente:

Figura 5.6 Pantalla del juego

Como se puede ver en la figura se puede distinguir los siguientes elementos:

- Un conjunto de Utensilios en la parte superior, son el conjunto de utensilios que se disponen, estos utensilios vienen cargados desde un XML que se carga al inicio del juego.
- Un mantel pequeño (a cuadros), habrá tantos manteles como comensales haya en la mensa, se han utilizado para dar una indicación a los usuarios de donde se encuentran los huecos donde se deben dejar los utensilios.
- Un utensilio “en movimiento”, es aquel que no está ni entre los situados en la parte superior ni está localizado en ningún hueco. Se crea cada vez que se selecciona un Utensilio de la parte superior, sólo puede haber uno, de manera que cada vez que se selecciona otro utensilio de la parte superior o de los situados en los huecos, se elimina el anterior utensilio “en movimiento”.

5.3.1.1 Ficheros de Configuración

En el juego se trabaja con ficheros XML, se puede distinguir entre dos tipos, dependiendo de su función.

- Fichero de carga de Utensilios, este fichero guarda los datos de los utensilios que se muestran en la parte superior del juego, en el fichero se guarda la información de los diferentes utensilios que se utilizan (tipo y nombre de su foto) así como la posición del primero (el resto van en línea) como la separación que deberá haber entre los diferentes utensilios.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="UtensiliosDisponibles">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="posicionPartidaX" type="xs:unsignedByte" />
        <xs:element name="posicionPartidaY" type="xs:unsignedByte" />
        <xs:element name="espacio" type="xs:unsignedByte" />
        <xs:element maxOccurs="unbounded" name="Utensilio">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="AssetName" type="xs:string" />
              <xs:element name="tipo">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="platoLlano"/>
                    <xs:enumeration value="tenedor"/>
                    <xs:enumeration value="cuchara"/>
                    <xs:enumeration value="cuchillo"/>
                    <xs:enumeration value="vaso"/>
                    <xs:enumeration value="platoHondo"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figura 5.7 Esquema XML de los utensilios disponibles

```
<?xml version="1.0" encoding="utf-8" ?>
<UtensiliosDisponibles>
  <posicionPartidaX>230</posicionPartidaX>
  <posicionPartidaY>30</posicionPartidaY>
  <espacio>10</espacio>
  <Utensilio>
    <AssetName>platoHondo</AssetName>
    <tipo>platoHondo</tipo>
  </Utensilio>
  <Utensilio>
    <AssetName>platoLlano</AssetName>
    <tipo>platoLlano</tipo>
  </Utensilio>
  <Utensilio>
    <AssetName>vaso</AssetName>
    <tipo>vaso</tipo>
  </Utensilio>
  <Utensilio>
    <AssetName>tenedor</AssetName>
    <tipo>tenedor</tipo>
  </Utensilio>
  <Utensilio>
    <AssetName>cuchara</AssetName>
    <tipo>cuchara</tipo>
  </Utensilio>
  <Utensilio>
    <AssetName>cuchillo</AssetName>
    <tipo>cuchillo</tipo>
  </Utensilio>
</UtensiliosDisponibles>
```

Figura 5.8 Ejemplo fichero XML utensilios

- Fichero de configuración de Huecos, en este fichero se establece el número de comensales, los utensilios que debe tener cada uno y los huecos donde que tienen para dejar estos utensilios (junto con los tipos de utensilios aceptados por estos huecos) y la orientación de estos, que servirá de información a la hora de dibujarlos.

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="Comensal" maxOccurs="unbounded" minOccurs="1">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="obligatorios">
            <xs:complexType>
              <xs:sequence>
                <xs:element maxOccurs="unbounded" name="Utensilio">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="platoLlano"/>
                      <xs:enumeration value="tenedor"/>
                      <xs:enumeration value="cuchara"/>
                      <xs:enumeration value="cuchillo"/>
                      <xs:enumeration value="vaso"/>
                      <xs:enumeration value="platoHondo"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="PosicionMantelX" type="xs:unsignedShort" />
          <xs:element name="PosicionMantelY" type="xs:unsignedShort" />
          <xs:element maxOccurs="unbounded" name="Hueco">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="CoordenadaCentralX" type="xs:unsignedShort" />
                <xs:element name="CoordenadaCentralY" type="xs:unsignedShort" />
                <xs:element name="tipo">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="platoLlano"/>
                      <xs:enumeration value="tenedor"/>
                      <xs:enumeration value="cuchara"/>
                      <xs:enumeration value="cuchillo"/>
                      <xs:enumeration value="vaso"/>
                      <xs:enumeration value="platoHondo"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:element>
                <xs:element name="orientacion">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="arriba"/>
                      <xs:enumeration value="abajo"/>
                      <xs:enumeration value="izquierda"/>
                      <xs:enumeration value="derecha"/>
                    </xs:restriction>
```

Figura 5.9 Fragmento esquema de lista de huecos

Ambos ficheros son cargados al inicio de la ejecución y si no resultan validos con el esquema XSD, la ejecución se interrumpirá.

5.3.1.2 La clase LectorXML.cs

El objetivo de esta clase es leer la información contenida en los XML de configuración, como se puede ver en la siguiente figura, la clase está formada por dos métodos estáticos, cada uno de ellos leerá un XML distinto siendo el método getHuecos() el que lea el XML referente a los huecos donde deberá el usuario dejar los utensilios y siendo getUtensilios() el método que leerá el XML referente a los Utensilios disponibles.

Cada método antes de leer el XML, lo validará con su esquema XSD correspondiente, si el XML no resulta validado, se informará al usuario de ello y se detendrá la ejecución del juego.

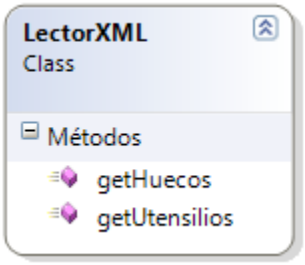


Figura 5.10 Clase LectorXML.cs

5.3.1.3 La clase Game1.cs, descripción general

Esta clase es la instanciada por Program.cs y hereda Game.cs (ver punto 4.1). En esta clase se tienen todos los métodos propios de XNA (Initialize, Update, Draw, etc.) que llamarán a los correspondientes del resto de clases de la aplicación (el resto de clases también tiene la estructura de una clase XNA para así simplificar el código).

Esta clase es la que se encarga de la "dinámica de juego", aquí es la que detecta cuando se coge un objeto o se deja, la que manda instanciar el objeto "en movimiento", etc, por eso debido a la importancia que tiene la interacción en todos estos procesos esta clase será distinta según el modo de interacción. No obstante hay ciertos elementos comunes:

- Los datos contenidos en los XML, es decir la lista de utensilios y la lista de comensales con sus huecos se guardan en las listas ListaUtensilios (una lista de objetos de la clase Utensilio) y la ListaPersonasHuecosUtensilio (lista de objetos PersonaParaComer) respectivamente.
    - La orden para cargar estas listas se da en el método Initialize() de Game1.
- Existe un objeto del tipo Utensilio (_utensilioEnDesplazamiento) que representa el utensilio "en movimiento.
- El constructor de la clase, en esta se selecciona tanto el tamaño de la pantalla, visibilidad del ratón, etc. En esta clase también es donde se instancia la clase KinectManager (o KinectManagerDosManos) que es la que se encarga de la interacción con Kinect.

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
    //se debe pasar el tamaño de la ventana donde moveremos el "cursor" del kinect
    graphics.PreferredBackBufferWidth = Game1.pantallaWidth;
    graphics.PreferredBackBufferHeight = Game1.pantallaHeigth;
    _kinectManager = new KinectManager(this, graphics.PreferredBackBufferWidth,graphics.PreferredBackBufferHeight);
    Components.Add(_kinectManager);
    graphics.IsFullScreen = true;
    this.IsMouseVisible = true;

    this._btnCorregir = new BtnCorregir();
}
```

Figura 5.11 Constructor de la clase Game1.cs

- En el método UnloadContent(), llamado al acabar la ejecución del juego, se llamará al método UnloadContent de KinectManager, donde entre otras cosas dependiendo del tipo de interacción se desinicializará Kinect y se pondrá la inclinación de su cámara a cero.

5.3.1.4 La clase Utensilio.cs

Es una de las clases principales del sistema, en ella está toda la información necesaria para describir un Utensilio (vaso, tenedor, plato, etc). Aunque se usarán Utensilios distintos (diferentes tipos y tamaños) se decidió utilizar la misma clase para describirlos y no tener un conjunto de clases (una para cada tipo de utensilio) que hereden de esta debido a que la única diferencia "real" entre estos utensilios es el tamaño de su representación gráfica, su funcionamiento dentro del programa es el mismo.

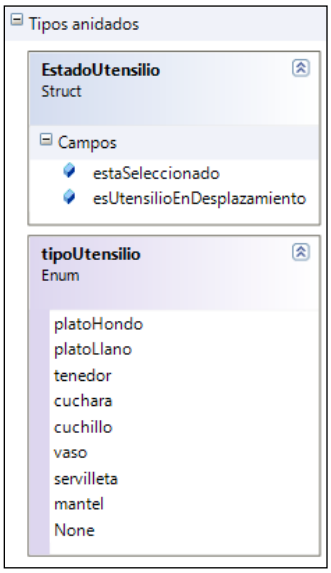


Figura 5.12 Tipos propios de la clase Utensilio.cs

Como se puede ver en la figura anterior, se han creado estructuras de datos propias dentro de la clase con el objetivo de hacer el código de la clase más legible. Estas dos estructuras son:

- Estructura EstadoUtensilio, esta estructura sirve para indicar si el utensilio es el utensilio en desplazamiento, es decir, el que el usuario ha seleccionado para colocarlo en un hueco. Esta estructura tiene dos campos booleanos:
    - estaSeleccionado, indica si el utensilio es el seleccionado por el usuario.
    - esUtensilioEnDesplazamiento, indica si el utensilio en ese momento se haya "agarrado" por el usuario, es decir si lo está moviendo en ese momento.
- Enumeración tipoUtensilio, guarda los diferentes tipos de utensilios que puede haber.

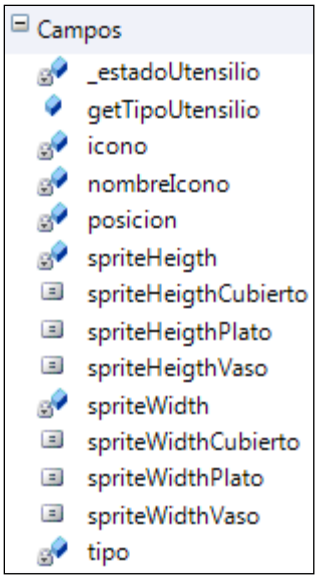


Figura 5.13 Campos de la clase Utensilio.cs

Como se puede ver en la figura, la clase Utensilio tiene muchos campos, detalladamente, las funciones de cada uno son las siguientes:

- Conjunto de constantes que indican el tamaño del sprite referente a la imagen de cada utensilio.
- getTipoUtensilio, es un diccionario que une el String que representa a un tipo de utensilio con el elemento Tipo (de la enumeración tipoUtensilio), este diccionario es accedido desde los métodos de la clase LectorXML para ligar el tipo de los utensilio leídos en el XML (se lee en formato String) con los elementos de la enumeración tipoUtensilio. El uso de diccionario permitirá simplificar el código además de proporcionar un acceso más rápido a los datos.
- nombreIcono e icono, representan el nombre de la imagen (tipo String) y la imagen que se va a dibujar (tipo Texture2D).
- spriteHeigth y spriteWidth, representan el tamaño del sprite que se corresponde con el utensilio.
- posición, guarda la posición donde se dibujará el sprite, se corresponde con su vértice superior izquierdo.
- tipo, guarda el tipo del Utensilio, es un elemento de la enumeración tipoUtensilio.

Todas estas propiedades son privadas a excepción de las constantes y del diccionario, para poder acceder desde el exterior a algunas de ellas se usan las siguientes propiedades:

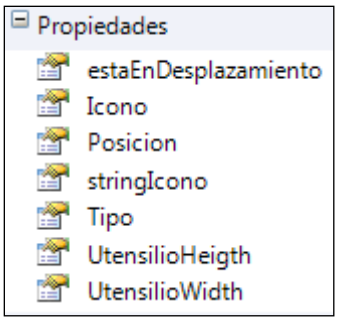


Figura 5.14 Propiedades de la clase Utensilio.cs

- estaEnDesplazamiento, el método get devuelve si está o no seleccionado por el usuario, es decir si lo está moviendo en ese momento.
    - su método set pone como true los dos elementos de _estadoUtensilio cuando se pasa como valor true, en cambio cuando se pasa false sólo modifica el elemento esUtensilioEnDesplazamiento. Esto se debe a que cuando un utensilio se marca como seleccionado para dejar de serlo tiene que ser borrado, por lo que no tiene sentido poder modificar este atributo.

El resto de propiedades devuelven el dibujo del utensilio, su posición, el nombre de la imagen de su utensilio, su tipo y su altura y anchura respectivamente.

Una vez explicados los diferentes campos y propiedades que definen a un objeto de la clase utensilio es conveniente explicar los diferentes estados por los que pasa un objeto de esta clase.

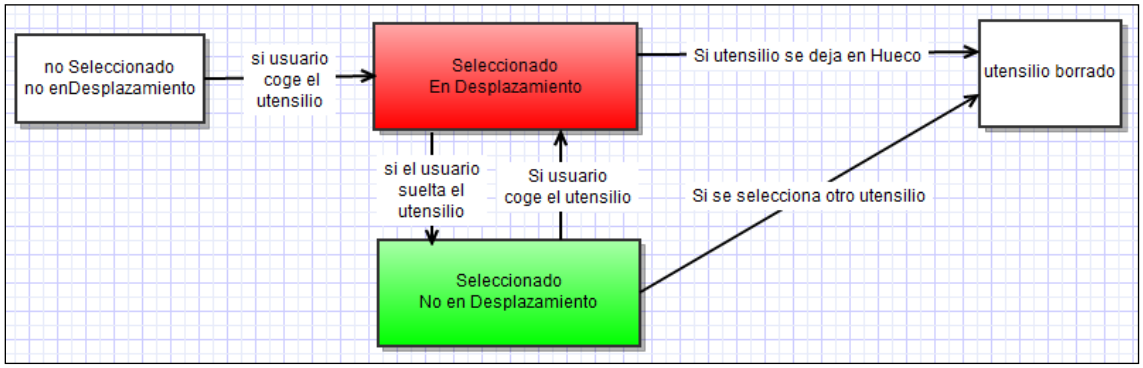


Figura 5.15 Estados de un Utensilio

El primer estado se corresponde a los utensilios de la parte superior de la pantalla o a los que se encuentran colocados ya en un hueco, cuando el usuario selecciona uno de ellos hará una copia de este utensilio que pasará al estado numero dos (seleccionado y en desplazamiento), una vez en este estado el usuario puede soltar el utensilio, que pasaría a estar en el estado tres (seleccionado pero no en desplazamiento) y volvería al estado dos si volviera a ser cogido, en el caso que el usuario seleccionara otro utensilio, el antiguo utensilio seleccionado sería borrado. Mientras se está en el estado dos es posible que el usuario suelte el utensilio en un hueco compatible con él, en ese caso el utensilio será borrado, aunque sus datos serán copiados en el hueco para que lo dibuje ahí.

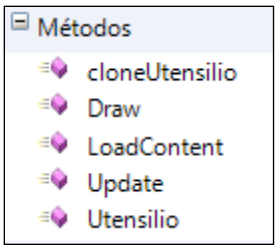


Figura 5.16 Métodos de la clase Utensilio.cs

La mayoría de los métodos de esta clase son copia de los métodos propios de XNA y son llamados desde sus métodos análogos de la clase Game1.cs.

- Utensilio(), es el método constructor, recibe como parámetros de entrada la posición, el tipo del utensilio y el nombre de la imagen del utensilio. Según su tipo calcula el tamaño del sprite a dibujar a partir de las constantes.
- LoadContent(), aquí se cargará el contenido gráfico, es decir, se obtendrá la imagen a dibujar a partir de su nombre.
- Update(), cuando desde la clase Game1 se considera que el objeto debe moverse (es decir cuando esta seleccionado y en desplazamiento), se le pasa la coordenada donde debe estar situado, esta coordenada se considera el punto central del sprite del utensilio por lo que a partir del tamaño del sprite se deberá calcular la posición del vértice superior izquierdo, cuyas coordenadas serán las que serán el nuevo valor del atributo posición de la clase.
- Draw(), este método es el encargado de dibujar el utensilio de acuerdo al valor del atributo posición en ese momento. Según el estado del utensilio para su dibujo será utilizado un color distinto:
    - Si el utensilio no está seleccionado ni en desplazamiento, se usará el color blanco.
    - Si el utensilio esta seleccionado y en desplazamiento se usará el color rojo.
    - Si el utensilio está seleccionado pero no en desplazamiento se usará el color verde.
- CloneUtensilio(), este método devolverá un objeto Utensilio que será copia exacta del actual objeto. Este método es llamado desde la clase Game1.cs cuando se detecta que el usuario ha seleccionado un nuevo Utensilio.

5.3.1.5 La clase PersonaParaComer.cs

Esta clase representa a cada una de las personas que va a comer, viene representado gráficamente por un mantel que servirá como ayuda al usuario para indicarle donde tiene que dejar los diferentes utensilios. Esta clase contendrá una lista con los Huecos que contiene y otra con los diferentes utensilios que debe poseer para ser correcta.

Como va a haber una lista de utensilios independientes (se ha planteado una mesa simple, es decir no tiene porqué haber más de un utensilio de cada tipo) se desea controlar cuales de los elementos imprescindibles se encuentran en el mantel, para ellos se ha creado la estructura UtensilioPresente que está formada por dos elementos:

- Tipo, que identifica el tipo de utensilio.
- Presente, que es un elemento del tipo booleano que indica si hay algún objeto del tipo ese dentro del mantel.

La clase PersonaParaComer tiene la siguiente estructura:

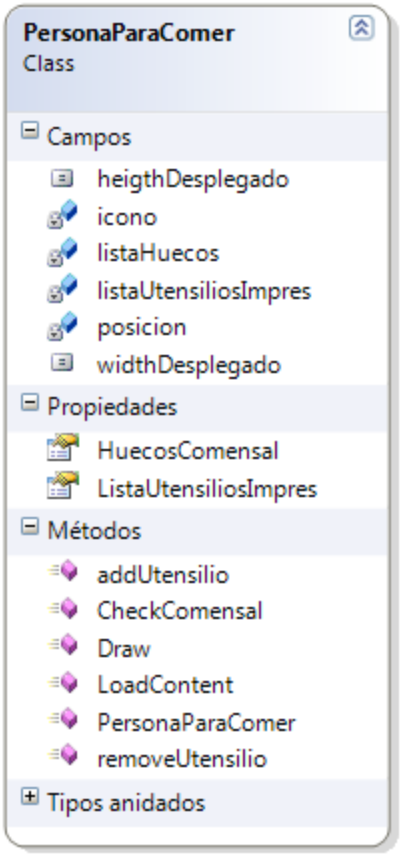


Figura 5.17 Clase PersonaParaComer.cs

La clase posee dos constantes que indican el tamaño que deberá tener el mantel.

Los atributos que definen la clase son:

- Icono, es la imagen del mantel.
- Posición, es la posición de la esquina superior izquierda del mantel.
- listaHuecos, es una lista de objetos de la clase HuecoUtensilio que representan los huecos de dentro del mantel donde hay que colocar los utensilio.
- listaUtensiliosImpres, es una lista de elementos de la estructura UtensilioPresente, en esta lista se lleva la cuenta de los utensilios que deben aparecer en el mantel y si están o no.

Los métodos de la clase son:

- PersonaParaComer, recibe una lista con los huecos que tiene así como otra con sus utensilios imprescindibles, también recibe la posición del mantel.

- AddUtensilio, al dejar un utensilio en un hueco hay que modificar la lista listaUtensiliosImpres para marcar el utensilio imprescindible correspondiente como presente.
- RemoveUtensilio, a coger un elemento colocado es como si se quitara también del mantel, por eso hay que cambiar la lista listaUtensiliosImpres para que, en el caso que el utensilio que hemos cogido sea el único de ese tipo en el mantel, marque como ese utensilio como no presente en la lista.
- LoadContent, carga la imagen del mantel.
- Draw, dibuja el sprite del mantel con el mantel.
- CheckComensal, este método devuelve true si todos los huecos están ocupados y si están presentes todos los Utensilios imprescindibles de la lista listaUtensiliosImpres.

5.3.1.6 La clase HuecoUtensilio.cs

Esta clase gestionara los datos y el comportamiento de cada Hueco. Un hueco es cada sitio donde se puede colocar un Utensilio, podrá aceptar uno o vario tipos de Utensilio, debiendo dibujar el utensilio cuando el real está sobre el (o cerca) o dibujarlo continuamente mientras se haya dejado un utensilio encima suya.

Por motivos de presentación es necesario establecer la orientación de los utensilios (hacia arriba, hacia abajo, etc.) al igual que con los tipos de Utensilio, se ha creado una enumeración que recoge los diferentes tipos de orientación así como un diccionario que liga el String que representa el tipo de orientación con la enumeración Orientación, este diccionario que se llama getOrientaciónHueco, al igual que ocurría con el diccionario de la clase Utensilio, es llamado desde los métodos de la clase LectorXML para ligar el String que indica la orientación en el XML con el elemento de la enumeración de la manera más simple y eficiente.

```
public enum Orientacion
{
    arriba,
    abajo,
    izquierda,
    derecha
}
```

```
public static Dictionary<String, HuecoUtensilio.Orientacion> getOrientacionHueco
{
    {"arriba", HuecoUtensilio.Orientacion.arriba},
    {"abajo", HuecoUtensilio.Orientacion.abajo},
    {"izquierda", HuecoUtensilio.Orientacion.izquierda},
    {"derecha",HuecoUtensilio.Orientacion.derecha}
};
```

Figura 5.18 Enumeración orientación y diccionario

Debido a que el Hueco deberá dibujar el Utensilio si lo tiene encima o si se lo han colocado, se han creado dos estructuras diferentes, una que guarde la información de lo que hay que pintar cuando el hueco está ocupado y otra cuando el hueco esté libre.

Cuando se pasa un Utensilio por un hueco compatible con él, es decir que uno de los tipos que acepta es el tipo del Utensilio, es necesario dibujar este utensilio en el hueco para indicar al usuario que puede dejar ahí el objeto, sólo será necesario guardar los datos que son necesarios para dibujar, es decir, el rectángulo del sprite, la imagen a dibujar y un booleano que indique si es necesario dibujar o no.

```
struct DibujoRectangulo
{
    public Rectangle rectangulo;
    public Boolean hayQueDibujar;
    public Texture2D iconoMarco;
}
```

Figura 5.19 Estructura DibujoRectangulo

Cuando el Hueco está Ocupado se debe guardar más datos que en la estructura anterior, como el tipo de utensilio que se guarda y el nombre de la imagen del Utensilio, esto se debe a que cabe la posibilidad que el usuario intente coger un elemento que está en un Hueco por lo que es necesario guardar la información necesaria para volver a instanciar un elemento del tipo Utensilio.

```
struct HuecoOcupado
{
    public Boolean estaOcupado;
    public Rectangle RectanguloIcono;
    public Texture2D icono;
    public string stringIcono;
    public Utensilio.tipoUtensilio tipo;
}
```

Figura 5.20 Estructura HuecoOcupado

Los campos que definen un Hueco son los siguientes:

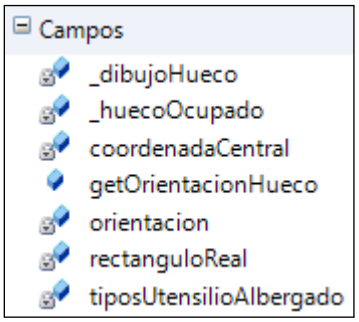


Figura 5.21 Campos de la clase HuecoUtensilio

- _dibujoHueco y _huecoOcupado representan a las estructuras descritas anteriormente.
- coordenadaCentral, es la coordenada central del Hueco, la que define su posición, debido a que en un mismo hueco puede haber diferentes tipos de utensilios con distintos tamaños, no se puede usar el vértice superior izquierdo como se usaban por ejemplo en los elementos de la clase Utensilio.
- getOrientacionHueco, es el diccionario ya explicado anteriormente.
- Orientación, es un elemento que marca la orientación del utensilio (pertenece a la enumeración antes explicada).
- TiposUtensilioAlbergado, lista de elementos Utensilio.Tipo que recoge los diferentes utensilios aceptados por el Hueco.

- RectánguloReal, para rotar la imagen, ajustarla a la orientación y que quede bien (que la coordenada central de ese rectángulo sea la mismo que el que se marca en la clase) es necesario coger un rectángulo desplazado hacia la izquierda o derecha o arriba o abajo, ese rectángulo es el que se guarda en las variables _dibujoHueco y _huecoOcupado, sin embargo es necesario guardar la información del rectángulo que ve el usuario, para calcular colisiones etc, ése rectángulo es el que guarda esta variable.

Como se puede observar en la figura la clase tiene métodos propios de XNA además de otros tantos propios:

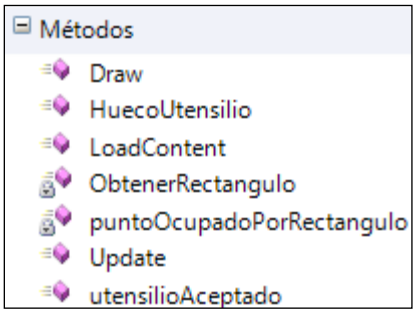


Figura 5.22 Métodos de la clase HuecoUtensilio

- HuecoUtensilio, es el constructor, recibe como parámetros de entrada su coordenada central, los tipos de utensilio aceptados y la orientación del hueco.
- puntoOcupadoPorRectángulo, devuelve verdadero si el rectángulo que contiene un Utensilio está sobre la coordenada central del hueco y su tipo pertenece al tipo de los aceptados por el hueco.
- ObtenerRectángulo, a partir de un tipo de utensilio, el punto central y la orientación, devuelve el rectángulo, que contendrá la imagen del utensilio, necesario para que al aplicar la rotación al dibujar quede el dibujo en el sitio deseado.
- utensilioAceptado, devuelve verdadero si el utensilio que se pasa como parámetro es compatible con el Hueco. De ser así se modifica la variable _huecoOcupado guardando los datos necesarios del Utensilio y poniendo su elemento estaOcupado a verdadero y el elemento _dibujoHueco.hayQueDibujar a falso para indicar que a partir de ese momento hay que empezar a dibujar los que marca la variable _huecoOcupado.
- Update, recibe de Game1 el utensilio que se encuentra en desplazamiento en ese momento, si en ese momento el Hueco se encuentra vacio y el utensilio esta encima del punto central y es compatible con el hueco se modificará la variable _dibujoHueco con la imagen del utensilio y se le calculará un rectángulo de acuerdo al tipo de orientación del hueco y tipo del Utensilio, finalmente se indicaría que es necesario dibujar lo estipulado por la variable poniendo el campo hayQueDibujar a true. En el caso de que el utensilio no sea compatible con el hueco o no este encima de la coordenada central de este se pondrá la variable hayQueDibujar a false para indicar que no se debe de dibujar.
- Draw, dibujará en el caso de ser necesario:
    - En el caso de estar ocupado el hueco dibujará lo establecido por la variable _huecoOcupado usando para ello el color blanco.
    - En el caso de que dibuje porque se halle un utensilio compatible sobre el hueco, se dibujara lo que aparece en la variable _dibujoHueco usando el color verde.

Véase que nunca se podrá dar el caso de dibujar lo contenido las dos variables a la vez. La rotación necesaria para cumplir con la orientación del hueco se calculará en este método también.

```
public void Draw(SpriteBatch spriteBatch)
{
    Vector2 origin = new Vector2(0, 0);
    float rotation;
    if (this.orientacion == HuecoUtensilio.Orientacion.abajo)
        rotation = MathHelper.Pi;
    else if (this.orientacion == HuecoUtensilio.Orientacion.derecha)
        rotation = MathHelper.Pi / 2;
    else if (this.orientacion == HuecoUtensilio.Orientacion.izquierda)
        rotation = 3 * MathHelper.Pi / 2;
    else
        rotation = 0;

    if (this._huecoOcupado.estaOcupado)
    {
        spriteBatch.Draw(this._huecoOcupado.icono, this._huecoOcupado.RectanguloIcono, null, Color.White, rotation, origin, SpriteEffects.None, 0f);
    }
    //si el hueco no está lleno
    else
    {
        if (this._dibujoHueco.hayQueDibujar)
            spriteBatch.Draw(this._dibujoHueco.iconoMarco, this._dibujoHueco.rectangulo,null,Color.Green,rotation,origin,SpriteEffects.None,0f);

    }
}
```

Figura 5.23 Método Draw de HuecoUtensilio

Al igual que en otras clases todos los atributos (menos constantes, definición de estructuras y el diccionario) son privadas, algunas de ellas es necesario poderles hacer referencia desde el exterior, para ello se han creado las propiedades que aparecen en la siguiente figura:

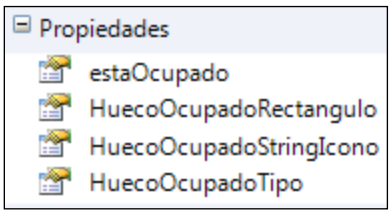


Figura 5.24 Propiedades de HuecoUtensilio

Estas propiedades devuelven un booleano diciendo si el hueco está ocupado, el rectángulo real (no el que se usa para dibujar), el nombre de la imagen y el tipo del utensilio que está colocado en el.

5.3.1.7 La clase BtnCorregir.cs

En la parte de la derecha de la pantalla aparecerá un botón, la forma de trabajar con él depende del tipo de interacción que se use. Dentro de esta clase se gestionarán los datos referentes a la representación gráfica del botón así como los métodos que gestionan la comprobación de fin del juego.

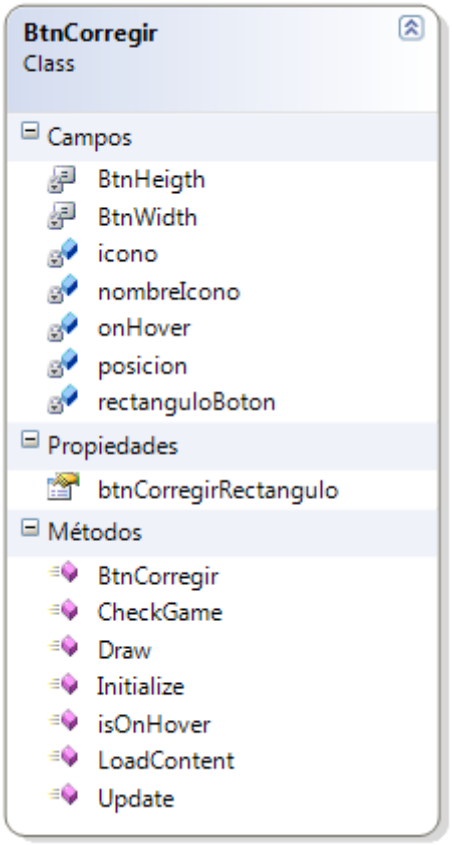


Figura 5.25 Esquema de la clase BtnCorregir

Como se puede ver en la figura, los atributos de la clase hacen todos referencia a la representación gráfica del botón.

- BtnHeigth y BtnWidth son dos constantes que hacen referencia al tamaño del botón.
- Icono y nombreIcono son la imagen y el nombre de la imagen del botón.
- rectanguloBoton, es el sprite donde se dibujará el botón.
- Posición es la localización del triangulo superior izquierdo del sprite.
- onHover, es una variable booleana que indica que el usuario ha puesto “la mano” encima del botón (varía según el tipo de interacción).

Entre los métodos de la clase están:

- BtnCorregir, es el contructor, no recibe ningún parámetro.
- Initialize, recibe como parámetros la posición y el nombre del icono a dibujar.
- LoadContent, carga la imagen a partir de su nombre, que está almacenado en la variable nombreIcono.
- Update, recibe la posición de la mano (o de las manos según el tipo de interacción) y llama a la función isOnHover que calcula si están sobre el botón, modificando la variable onHover.
- Draw, dibuja el botón, usando el color verde en el caso de que onHover sea verdadero y blanco en caso contrario.

- CheckGame, se le llama cuando se quiere comprobar que si se ha completado la misión correctamente, recibe como parámetro la lista de comensales y la recorre uno a uno llamando a su función CheckComensal (ver punto 4.3.1.5) si todos los comensales están completados correctamente la función devolverá true.

5.3.1.8 La clase BotonNoFin.cs

Si la comprobación de final del juego no es correcta es necesario mostrar al usuario alguna señal que lo indique. La señal consistirá en una imagen que aparecerá durante tres segundos y luego desaparecerá. La clase BotonNoFin.cs gestionará todo esto.

Figura 5.26 Imagen de BotonNoFin.cs

Como se puede ver en la siguiente figura, esta clase lo único que gestiona es cuando mostrar o no la imagen de la figura anterior. Sus propiedades están todas relacionadas con la representación gráfica:

- Heigth y Width, son dos constantes que indican el tamaño de la imagen.
- Icono, es la imagen a mostrar.
- Posición, se corresponde con la posición de la esquina superior izquierda de la imagen a mostrar.
- Mostrar, es una variable de tipo booleano que indica si hay que mostrar la imagen o no.
- Tiempo, guarda dentro de una estructura DateTime el momento en el que se empezó a mostrar la imagen, es necesario guardarlo ya que, como se ha dicho antes, la imagen va a estar visible sólo un tiempo determinado.

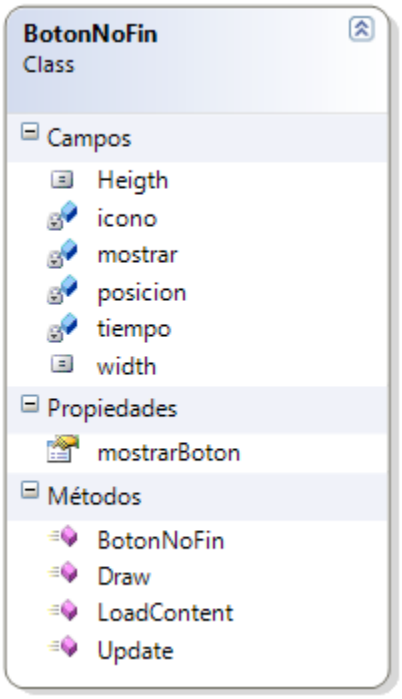


Figura 5.27 Esquema clase BotonNoFin

Para instanciar la clase se llamará a su constructor, al que se le pasará como parámetro de entrada la posición, el método LoadContent cargará la imagen y el método Draw la dibujará en función de si el valor de mostrar es verdadero o falso. El método Update lo que hace es comprobar la hora y fecha actual con la hora y fecha de cuando se puso la variable mostrar a true (variable tiempo), en el caso de que el tiempo transcurrido fuera superior a tres segundos, cambiaría el valor de la variable a false para que se dejase de dibujar la imagen.

Para cambiar el valor de la variable mostrar a true se hace uso de la propiedad mostrarBoton, esta al pasarle un valor true, modificará la variable mostrar para empezar a mostrar la imagen, además de guardar en la variable tiempo la fecha y hora de ese momento. Esta propiedad es llamada después de que el método CheckGame devuelva false.

5.3.2 Métodos de interacción

Una de las partes más importantes del desarrollo de la aplicación es encontrar el mejor modo de interacción. Es necesario encontrar una forma de interacción lo más intuitiva posible y que requiera el mínimo tiempo de adaptación, contra más sencillo sea el manejo más atractiva le resultará la aplicación al usuario.

La forma de interacción ideal sería registrando los movimientos de los dedos, de manera que al cerrar la mano se cogiera el utensilio y al abrirla se dejara el utensilio. Sin embargo este modo de interacción es imposible debido a las limitaciones del SDK que no reconoce los dedos.

Otra opción de interacción barajada fue mediante el uso del gesto Push (ver capitulo 3), sin embargo, aunque se trata de un movimiento bastante intuitivo, su reconocimiento por parte del sensor no lo es,

teniéndolo que hacer de una manera que a un usuario inexperto consideraría como demasiado forzada. Es por eso que esta forma de interacción también se rechazó.

Finalmente se decidió desarrollar dos tipos distintos de interacción:

- Utilizando comandos de voz, con diferentes comandos de voz para coger un objeto, dejarlo, acabar el juego, etc. haciendo uso del reconocimiento de voz que ofrece Kinect.
- Utilizando las dos manos para coger un objeto, si las dos manos rodean el objeto se coge, en el momento que se separan se deja.

5.3.2.1 Interacción mediante el uso de comandos

Para este tipo de interacción se usará solamente una mano, al ponerla encima de utensilio o hueco ocupado y decir la palabra “take” se cogerá el utensilio, para soltarlo será necesario decir la palabra “leave”. Una vez se crea que se ha acabado se dirá “finish” y el sistema comprobará que se ha completado la misión correctamente, si no es así se mostrará la imagen de error, si es correcto se mostrará al usuario la pantalla de fin del juego. Para reiniciar el juego será necesario decir “restart”.

5.3.2.1.1 La clase KinectManager.cs

La clase que controla el sistema se llama KinectManager y está basada en las explicadas en el capitulo anterior. En esta clase se realizará un tracking de la mano a la vez que intenta reconocer las palabras que dice el usuario.

El sistema de reconocimiento de voz consiste en que Kinect va “escuchando” y cuando reconoce una palabra que responde a un patrón salta un evento.

Las palabras a reconocer son:

- Take, para coger un objeto.
- Leave, para dejar un objeto.
- Finish, para comprobar si se ha acabado la misión.
- Restart, para volver a empezar la misión.

Una vez se ha iniciado el reconocedor de habla podrán saltar tres tipos diferentes de eventos,

- Hipótesis de nueva palabra, salta cuando se cree que se ha detectado una palabra.
- Palabra detectada, cuando se ha detectado una palabra.
- Palabra rechazada, cuando se había hecho una hipótesis sobre una palabra, pero finalmente no era ninguna de las palabras reconocidas.

Para el tracking de la mano usara la lectura de Joints con la misma técnica que la usada en el tema anterior, además se ha añadido la posibilidad de poder cambiar de jugador cuando este salga de la pantalla y

que mientras esté el resto de jugadores que haya en el área de juego no interfieran. Para conseguir esto se guarda el Id del primer jugador detectado (en la variable skeletonIdJugador), cuando se detecte un nuevo frame sólo se trataran los esqueletos que tengan el mismo Id (es decir, los esqueletos que se corresponden a ese jugador), si pasa un tiempo (500 milisegundos) sin detectar ningún esqueleto con ese Id, se considerará que el jugador ha salido del área de juego, por los que el Id se pondrá a 0 (ningún esqueleto tiene Id 0) y se sobrescribirá por el valor del Id del primer esqueleto que se detecte.

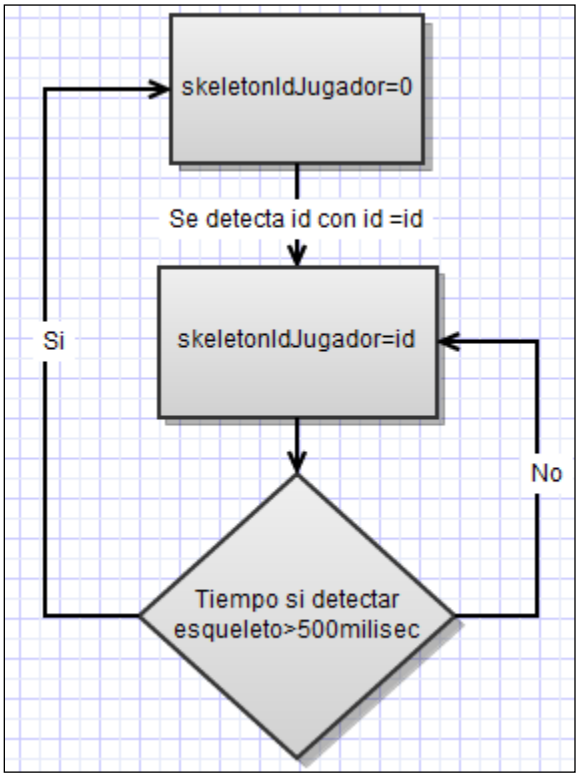


Figura 5.28 Diagrama de flujo del atributo skeletonIdJugador

Aunque implementa algunos métodos de XNA, esta clase funciona principalmente con los eventos propios de Kinect y para comunicarse con la clase Game1 utiliza también eventos, esta vez propios, estos eventos se corresponden con las órdenes que da el usuario mediante palabras.

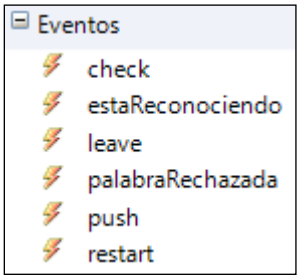


Figura 5.29 Eventos de la clase KinectManager

El caso de los eventos estaReconociendo y palabraRechazada son utilizados para poder hacer juego más dinámico. El gran problema del reconocimiento de palabras es que desde que se dice la palabra hasta que se detecta pasa por lo menos dos segundos, de cara al usuario final esto es mucho tiempo, dando la sensación de una falta de control sobre la aplicación. El evento estaReconociendo es lanzado cuando se detecta la hipótesis de palabra detectada (que ocurre mucho antes que el de palabra reconocida) de esta manera cuando se reciba este evento se podrá realizar una acción que indique al usuario que se está reconociendo su palabra, haciendo que el tiempo de espera parezca menor, esta acción se parará cuando se detecte una palabra (se lanza cualquiera de los eventos que reconocen las órdenes de un usuario) o cuando se rechace una palabra (palabraRechazada).

Los métodos de la clase son los siguientes:

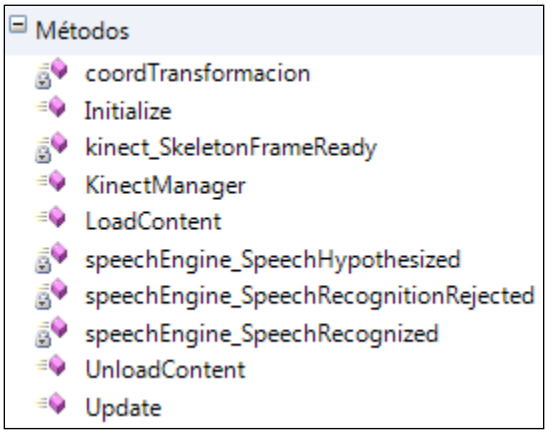


Figura 5.30 Métodos de la clase KinectManager

- El constructor KinectManager recibe como parámetros los pixel de ancho y de largo de la pantalla del juego, estos datos son necesarios para calcular la posición de la mano en la pantalla.
- El método Initialize, inicia el Kinect, tanto la parte relativa a la detección de esqueleto (ver capitulo anterior) como la parte relativa al reconocimiento de palabras (iniciando el flujo de audio, seleccionando las palabras a detectar, etc).
- UnloadContent, en este método se cerrarán los flujos de audio y se parará el reconocedor de voz, así como se desinicializará Kinect.

```
public void UnloadContent()
{
    kinectSource.Stop();
    speechEngine.RecognizeAsyncCancel();
    speechEngine.RecognizeAsyncStop();
    kinectSource.Dispose();
    kinect.NuiCamera.ElevationAngle = 0;
    kinect.Uninitialize();
}
```

Figura 5.31 Método UnloadContent de KinectManager

- Kinect_skeletonFrameReady, método parecido a sus análogos explicados en el capítulo anterior con el añadido de la detección de salida/entrada de nuevos esqueletos explicado anteriormente. En esta clase se obtendrá la posición del Joint de la mano del jugador y se adaptará su posición al tamaño de la pantalla de juego con la ayuda del método coordTransformacion.
- speechEngine_SpeechHypothesized, método que se ejecuta cuando se detecta el evento de hipótesis de una palabra, lanza el evento estaReconociendo.
- speechEngine_SpeechRecognitionRejected, método que se ejecuta cuando se rechaza una palabra, lanza el evento palabraRechazada.
- speechEngine_SpeechRecognize, este método es lanzado si reconoce que el jugador ha dicho una palabra de las que se han indicado, según la palabra dicha lanzara un evento u otro:
    - take -> evento Push.
    - leave -> evento Leave.
    - Finish -> evento check.
    - Restart -> evento restart.

Aparte de mediante los eventos es necesario transmitir al exterior otros datos como son la posición actual de la mano y el rectángulo del sprite de esta (cuyo tamaño está definido en constantes de esta clase. Para acceder a estos datos se han creado las siguientes propiedades:

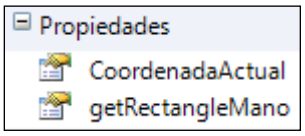


Figura 5.32 Propiedades de la clase KinectManager

Coordenada actual devuelve un Punto con la posición de la mano (en la pantalla de juego) y getRectangleMano calcula el rectángulo del sprite de la mano tomando la coordenada de la mano como punto central del rectángulo.

5.3.2.1.2 La clase Game1.cs

La clase Game1.cs es una clase que hereda de la clase Game de XNA, esta clase es la que lleva todo el peso de la aplicación ya que es la clase que sirve de unión entre las demás. Como ya se explicó en el punto 4.3.1.3 esta clase depende mucho del tipo de interacción.

Los métodos de la clase Game1 se pueden dividir en dos grandes grupos:

- Métodos propios de XNA.
- Métodos no propios, estos métodos son utilizados bien para simplificar el código o bien para recoger los eventos lanzados por la clase KinectManager.

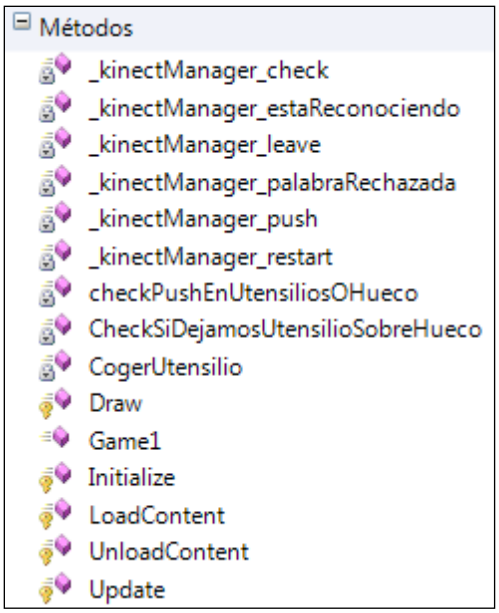


Figura 5.33 Métodos de la clase Game1.cs

Métodos no propios de XNA.

Algunos atributos importantes:

- Variable Booleana finDelJuego, cuando el usuario comprueba si ha completado correctamente y la respuesta es correcta, es necesario indicar al programa si mostrar la pantalla de fin del juego o no, esta variable es la que nos dice si el juego ha acabado o no.
- Variable Booleana isPush, no se desea que cuando un usuario está moviendo un utensilio aparezca el icono de la mano del jugador, por eso cuando un utensilio está en desplazamiento esta variable toma el valor true para indicar que no es necesario dibujar el icono de la mano.
- Variable Booleana reconociendoPalabra, para dar al usuario la señal de que se ha escuchado lo que dice y que se está reconociendo la palabra, mientras esté reconociendo la palabra se cambiará el icono que representa la mano del jugador.

Se han creado dos métodos que ayudarán a simplificar el código y a hacerlo más legible:

- CogerUtensilio, Este método comprueba si cuando se ha dado la orden “take” se estaba sobre un utensilio o Hueco ocupado. Lo primero que comprobará es que no se tenga cogido ningún utensilio en ese momento, si no es así se comprobará si se está sobre el utensilio seleccionado (si es que lo hay) y si no se llamará al método checkPushEnUtensilioOHueco, si este método devuelve true significará que se tiene un nuevo utensilioEnDesplazamiento que se está desplazando, por eso se pondrá la variable isPush a true y la variable reconociendoPalabra a false.
- checkPushEnUtensilioOHueco, método que devuelve true si se está sobre algún elemento de la lista de utensilios que se encuentra en la parte superior o de la lista de Huecos, si la respuesta es afirmativa, se hará un clon del Utensilio (método cloneUtensilio en el caso de estar sobre un utensilio o instanciar un nuevo Utensilio a partir de los datos de la estructura _huecoOcupado de la clase HuecoUtensilio en el caso de estar sobre un Hueco ocupado.

- CheckSiDejamosUtensilioSobreHueco, este método comprueba si el utensilio que se está dejando se deja sobre un HuecoCompatible en el caso de ser así se llamará al método addUtensilio de la clase PersonaParaComer (que llevará la cuenta de los utensilios están en los Huecos de cada comensal). Además en el caso de que se deje el utensilio en un hueco compatible el método borrara el actual utensilioEnDesplazamiento.

Los métodos que gestionan los eventos lanzados por KinectManager son los siguientes:

- _kinectManager_estaReconociendo, si se está desplazando un utensilio o se está sobre un utensilio o hueco ocupado, la variable booleana reconociendoPalabra se pondrá a true.
- _kinectManager_palabraRechazada, como ya no se está reconociendo una palabra, la variable booleana reconociendoPalabra se pondrá a false.
- _kinectManager_push, si no se ha acabado el juego, se llamará al método CogerUtensilio y se pondrá la variable booleana reconociendoPalabra a false.

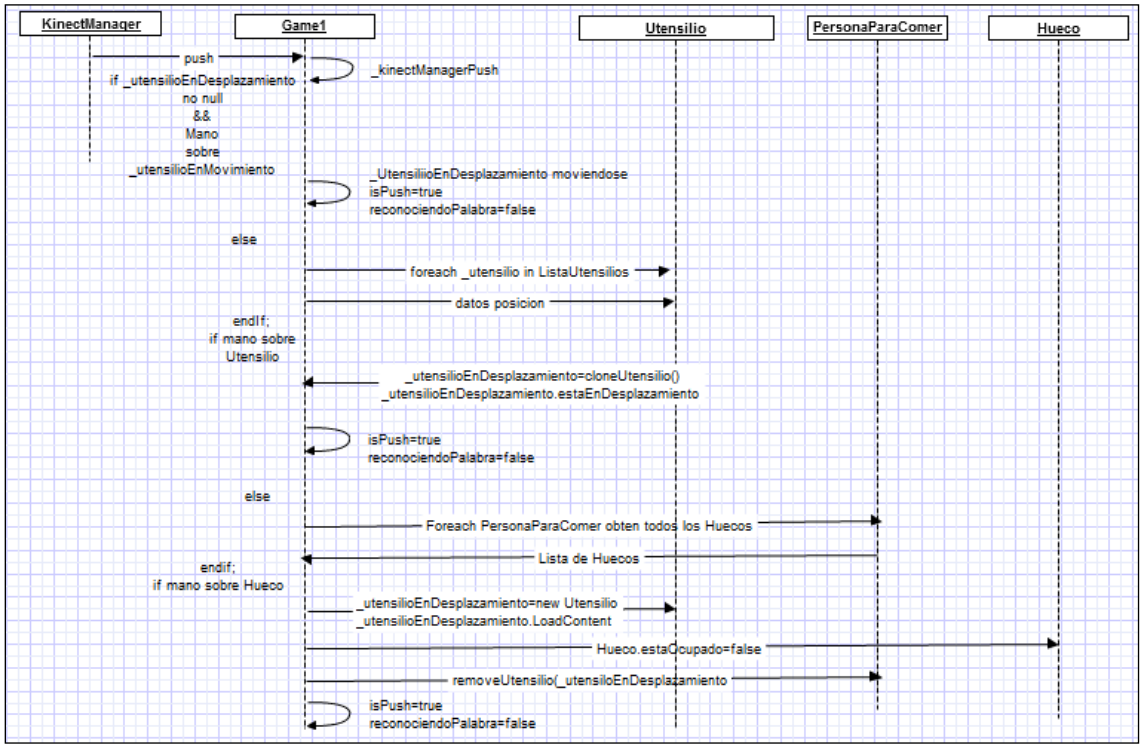


Figura 5.34 Diagrama de Secuencia acción coger elemento

- _kinectManager_leave, si existe un utensilio seleccionado que se haya en desplazamiento en ese momento, se suelta el utensilio y se llama al método checkSiDejamosUtensilioSobreHueco, que comprobará si el utensilio se ha dejado sobre un hueco compatible con este. En todas las circunstancias se pondrán las variables isPush y reconociendoPalabra a false.

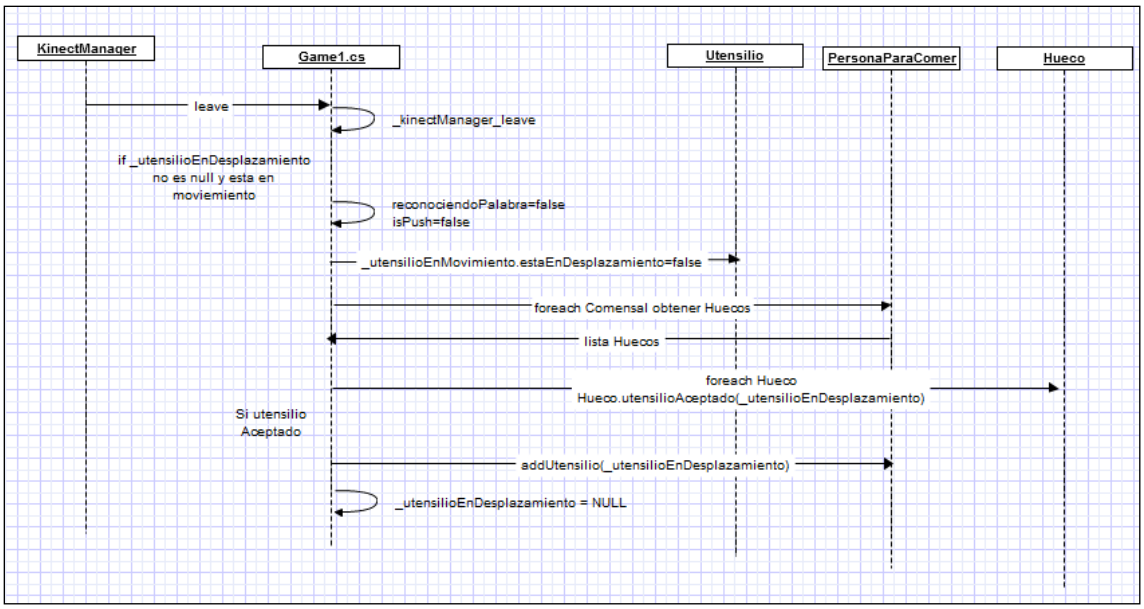


Figura 5.35 Diagrama de secuencia acción dejar Elemento

- _kinectManager_check, se llamará al método CheckGame de la clase BtnCorregir, en el caso de que no haya resuelto correctamente la misión se llamará al método mostrarBoton de la clase BotonNoFin para que se muestre la imagen de error.

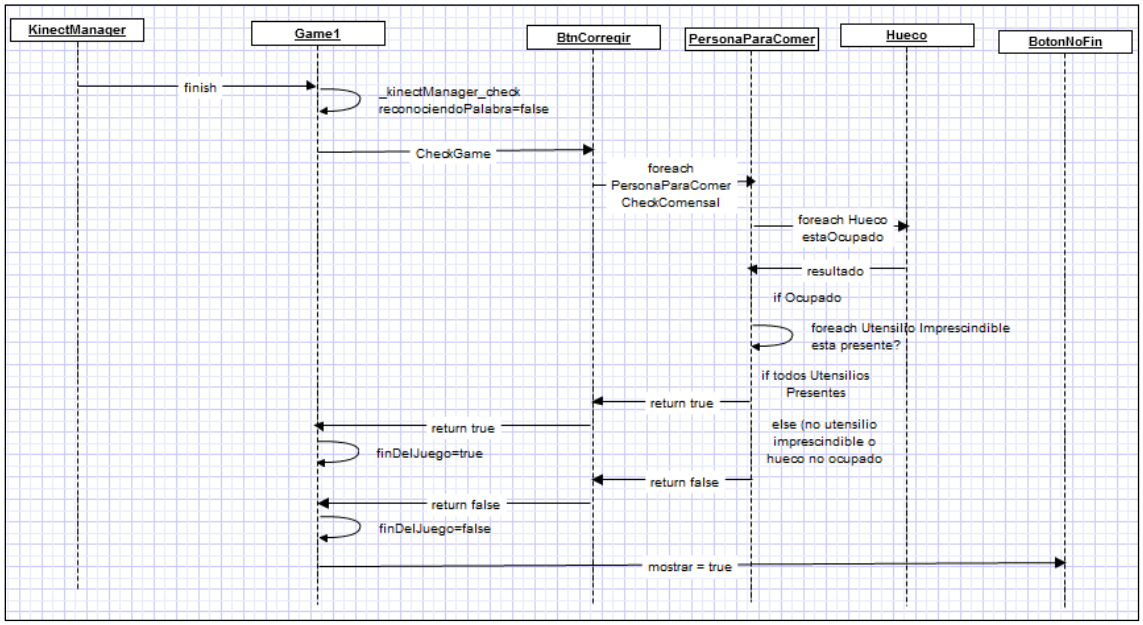


Figura 5.36 Diagrama de secuencia acción check

- _kinectManager_restart, si el usuario ha acabado correctamente el juego (finDelJuego igual a true), borrará el utensilio en desplazamiento y volverá a cargar la lista de comensales (con sus huecos) de un fichero XML (será necesario llamar a los métodos LoadContent de los comensales y de los huecos).

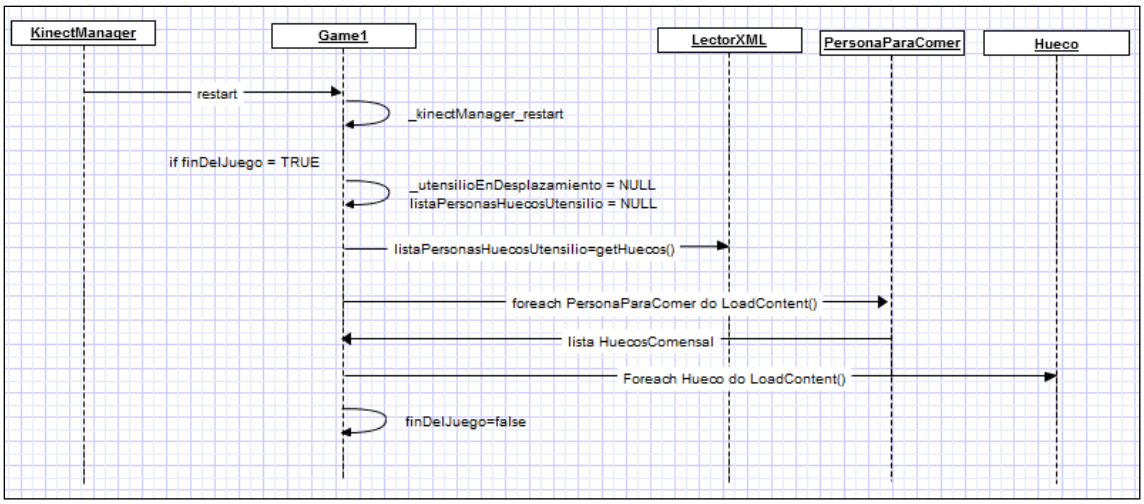


Figura 5.37 Diagrama de secuencia acción restart

Los métodos propios de XNA son los siguientes:

- Initialize, aquí cargan las listas de utensilios y comensales así como se instancian los botones de chequear y el de BotonNoFin, también se asocian métodos que recojan los eventos lanzados desde KinectManager.
- LoadContent, en este método se recorrerán todos los objetos de la clase Utensilio, PersonaParaComer, Hueco, BtnCorregir y BotonNoFin y se llamará a sus correspondientes métodos LoadContent..
- Update, en el caso de haber un Utensilio que se esté desplazando se actualizará su posición en función de la posición que de la propiedad de KinectManager CoordenadaActual. También se llamará al método Update de cada Hueco, ya que es posible que el utensilio que tenemos en desplazamiento en ese momento este encima del hueco por lo que sería necesario dibujar una copia del Utensilio en el Hueco. También se llamara a los métodos Update de BotonNoFin (para que compruebe si se ha pasado el tiempo de mostrar la imagen) y de BtnCorregir.
- Draw, Esta clase dibujara los diferentes elemento:
    - Siempre dibujará el fondo de pantalla y llamará al método Draw de BtnCorregir.
    - En el caso de que finDelJuego fuera verdadero, pintará la imagen de fin de juego.
    - En el caso de que sea falso finDelJuego, llamará a los métodos Draw de todos los Utensilios de la lista listaUtensilios, a los de las clases PersonaParaComer y a todos sus Huecos. También llamaría al método Draw del _utensilioEnDesplazamiento, si es que lo hubiera.
    - En el caso de que isPush fuera falso se dibujaría la mano del usuario, usando el rectángulo que ofrece la propiedad getRectangleMano de KinectManager.
    - Si reconociendoPalabra es true, se dibujará una estrella en vez de la mano del usuario (usando también el rectángulo de la propiedad getRectangleMano).

5.3.2.2 Interacción mediante el uso de las dos manos

En este tipo de interacción se utilizarán las dos manos, el modo de coger un Utensilio será necesario cogerlo con las dos manos, es decir, que los sprites de las manos estén en contacto con el sprite del utensilio. Para dejar el utensilio en un principio bastaría con separar las manos de manera que no estén en contacto con el rectángulo del objeto, pero para evitar que sea fácil que se suelte involuntariamente el objeto (hay que recordar que en el reconocimiento del Skeleton la posición de los Joints de un frame a otro experimenta cierta vibración) se tomará como sprite otro que será el doble de grande que el sprite real del utensilio.

Para comprobar si se ha realizado correctamente la misión se deberán colocar las dos manos sobre la imagen del botón corregir (situado a la derecha de la pantalla), si no se ha finalizado correctamente se mostrará la imagen de error y si no la del fin del juego. Para reiniciar el juego será necesario volver a poner las dos manos sobre el botón de la derecha.

5.3.2.2.1 La clase KinectManagerDosManos.cs

Esta clase realizará el tracking del esqueleto y obtendrá las coordenadas adaptadas al área de juego de los Joints de las manos. Realizará el tracking de manos de la misma manera que la clase KinectManager y tendrá el mismo sistema para calcular cuando un jugador ha abandonado el área de juego.

En esta clase además se desarrollo un sistema para “reducir el área de juego”, de manera que el usuario sólo con levantar la mano llegara hasta la parte superior de la pantalla. El tamaño de la pantalla sería el doble del tamaño de la distancia del cuello a la cintura (pixeles), el tamaño de este rectángulo variaría según el usuario de acercara o se alejara. El punto central de este cuadrado sería el cuello del jugador. Si por la parte superior el cuadrado se saldría del campo de juego “real”, este cuadrado sería cortado.

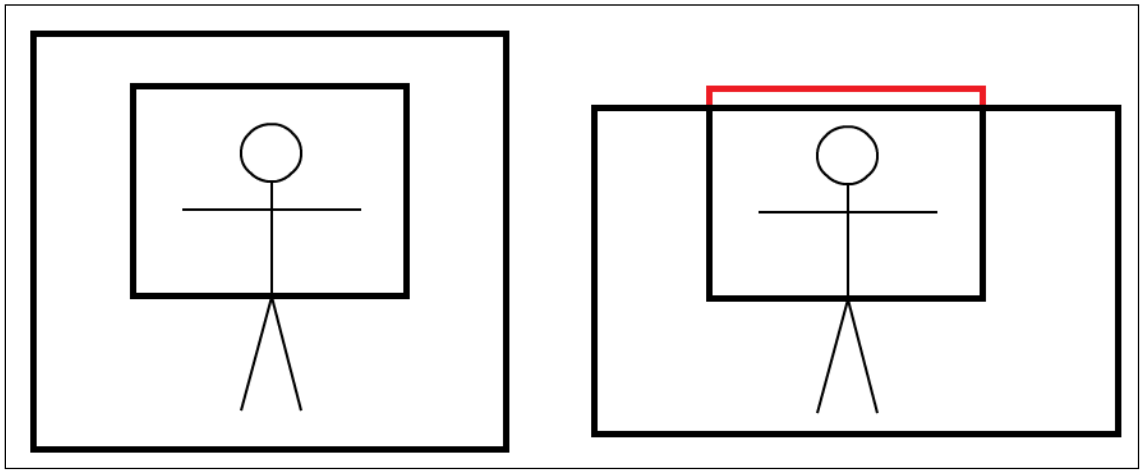


Figura 5.38 Efecto reducción del área de juego

Finalmente tras probar esta posibilidad ya que al reducir el “área de juego” lo que realmente se está haciendo es asignar a movimientos pequeños del brazo desplazamientos mayores en la pantalla, debido a esto

la vibración que se produce en el reconocimiento del esqueleto (el SDK no es lo suficiente potente todavía) se ve aumentada mucho, haciendo imposible la interacción con el juego.

Desde el exterior, al igual que la clase KinectManager, se deseará acceder a la posición de las manos y a los rectángulos (área donde se dibujara el icono de la mano) de estas, al igual que con la clase KinectManager la posición del Joint se corresponderá con la coordenada central del rectángulo, también es necesario saber el punto medio entre las dos manos (para dejar un utensilio). Para acceder desde el exterior se han desarrollado las siguientes propiedades:

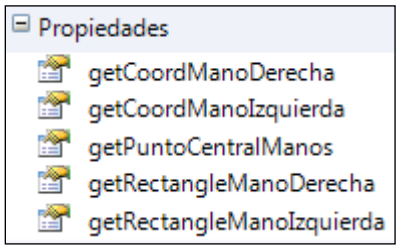


Figura 5.39 Propiedades de la clase KinectManagerDosManos.cs

5.3.2.2.2 La clase Game1.cs

Al igual que en el modo de interacción por voz esta clase es la que lleva el peso de la aplicación. En este caso también se puede dividir sus métodos en dos grandes grupos, los métodos propios de XNA y los otros.

- Tanto el constructor como el método Initialize son iguales que los del otro modo de interacción, con la diferencia que en este último no se asocian los eventos a métodos (KinectManagerDosManos no lanza eventos).
- LoadContent, aquí se cargarán las imágenes del icono de la mano, la imagen de fondo y la imagen de fin del juego, también se llamará a los métodos LoadContent del resto de clases (Utensilio, PersonaParaComer, BotonNoFin, BtnCorregir, etc).
- Draw, se irán dibujando elementos en el siguiente orden:
    - Imagen de fondo y se llamará al método Draw del objeto BtnCorregir.
    - Si no se ha acabado el juego:
        - Se llamará al método Draw de cada objeto PersonaParaComer y de todos sus objetos Hueco.
        - Se llamará al método Draw de todos los objetos Utensilio de la lista listaUtensilios.
        - Si existe elemento en desplazamiento se llamará a su método Draw.
        - Se llama al método Draw del objeto BotonNoFin.
    - Si ha acabado el juego se dibujará la imagen de fin de juego
    - Se dibujarán los dos iconos correspondientes a las dos manos, a diferencia de la interacción con órdenes de voz, en esta versión los iconos de las manos son siempre visibles.

- Método Update, en este método se lleva toda la lógica del juego, su funcionamiento se puede ver en el diagrama de flujo de la figura siguiente.

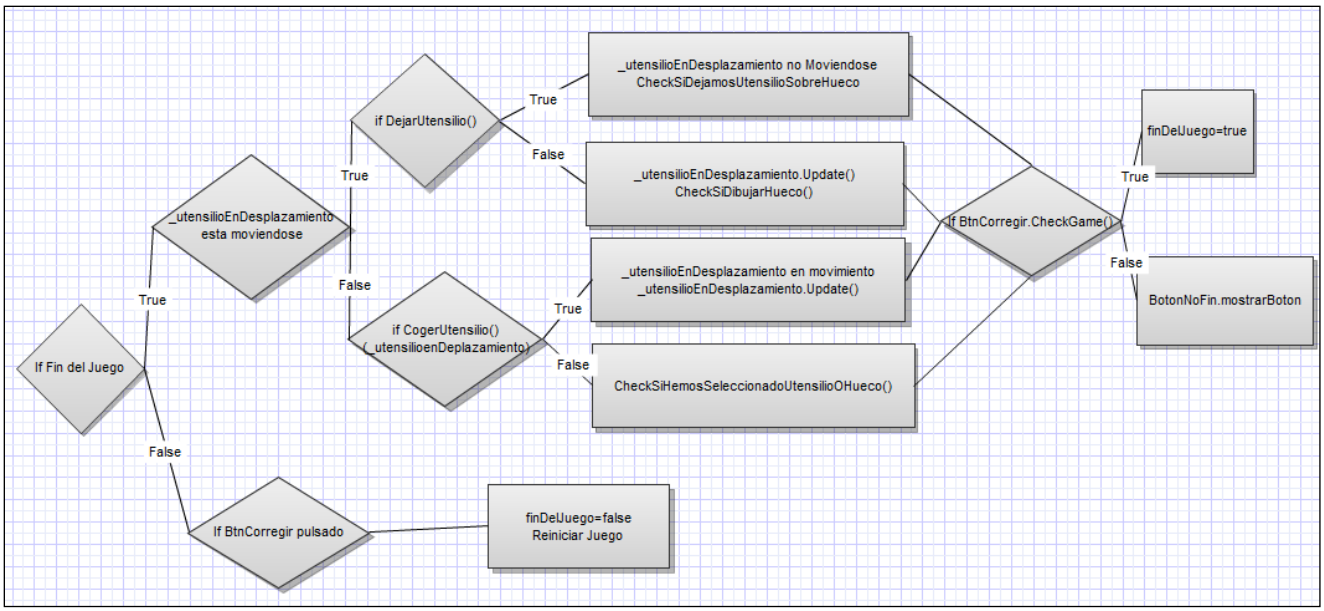


Figura 5.40 Diagrama de Flujo Game1.Update()

El resto de métodos, a los que se hace referencia desde las clases propias de XNA son los siguientes:

- CogerUtensilio, como su nombre indica, este método devuelve true si las manos están rodeando el sprite del utensilio. Para calcular esto se obtiene el rectángulo del Utensilio y los de los iconos de la mano, si los dos rectángulo de los iconos están en contacto (hay una intersección) con el rectángulo del utensilio, entonces se considera que se ha cogido el utensilio, y se devuelve true;
- DejarUtensilio, el método devolverá true si se ha dejado de coger un utensilio. Es prácticamente lo contrario de CogerUtensilio, es decir, que cuando los rectángulos de las manos dejen de hacer intersección con el rectángulo del utensilio se considerará que se ha dejado el utensilio. Esto es correcto, pero hay que añadir que ahora el rectángulo del utensilio no es el rectángulo "real" del utensilio, sino que es uno cuyo centro es el mismo pero la longitud de sus lados es doble. De esta manera se intenta evitar que el usuario suelte el utensilio sin quererlo (debido a la "vibración" de los Joints en el Skeletal tracking).
- SeHaSeleccionadoHueco, este método comprueba si se ha seleccionado el Utensilio que se encuentra colocado en un Hueco, de ser así el método devolverá true. El método lo primero que hace es comprobar si el Hueco está ocupado (propiedad estaOcupado), si es así, comprueba que los rectángulos de las manos están en contacto con el del utensilio que está en el hueco (se calcula a partir de los datos guardados en HuecoOcupado).
- CheckSiDejamosUtensilioSobreHueco, comprobará si al dejar un utensilio se ha dejado sobre un hueco, devolviendo true en ese caso. Para ello recorrerá todos los huecos de la mesa hasta que en uno de ellos el método Hueco.utensilioAceptado() devuelva true. En ese momento añadirá el utensilioEnDesplazamiento al comensal de quien es el hueco (método addUtensilio) y después borrará el objeto utensilioEnDesplazamiento.

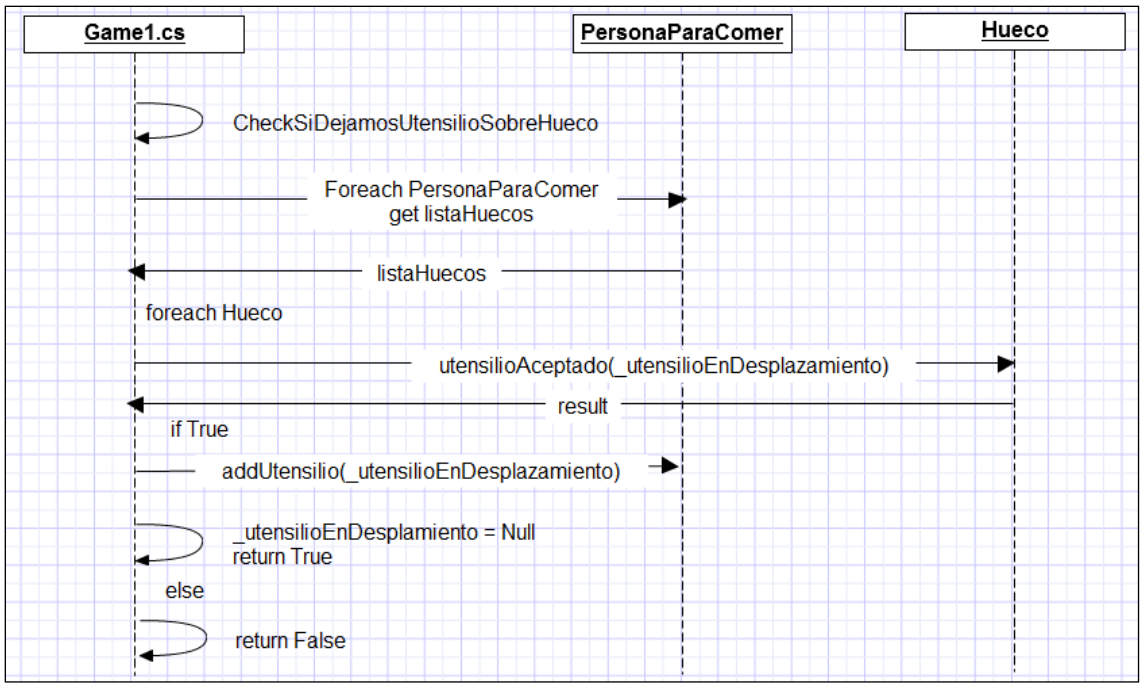


Figura 5.41 Diagrama de Secuencia de CheckSiDejamosUtensilioSobreHueco

- CheckSiDibujarHueco, este método recorre todos los huecos de los comensales llamando al método Update de cada Hueco pasándole el _utensilioEnDesplazamiento para que en el caso de que este encima de un Hueco y este sea compatible con el tipo del utensilio se dibuje el utensilio en el hueco (y así ayudar al usuario).
- CheckSiHemosSeleccionadoUtensilioOHueco, esté método hará un recorrido por todos los Utensilios y los Huecos para comprobar si ha seleccionado un utensilio (se llama al método CogerUtensilio) o un Hueco (método SeHaSeleccionadoHueco), en el caso de ser así se instanciará un nuevo objeto del tipo Utensilio que será el _utensilioEnDesplazamiento. Las formas de instanciarlo dependerán de si se ha seleccionado un utensilio o un Hueco:
    - Si es un Utensilio, se llamará al método cloneUtensilio() de este.
    - Si es un Hueco, con la información del utensilio albergado que está guardada en la clase se instanciará un nuevo objeto Utensilio. Será necesario llamar al método RemoveUtensilio del objeto PersonaParaComer correspondiente.

5.3.2.2 Pruebas con usuarios de los métodos de interacción

Se planteo una prueba en la que se trabajaría con tres niños de diferentes edades, dos niños de 11 años (uno con ASP) y otro de 6 años. La mecánica de la prueba fue la siguiente, entraban los niños de uno en uno, de manera que al entrar no tenían ninguna noción de cómo funcionaba el juego. Se le dejaba experimentar un poco sin darles ninguna información para ver cómo son sus primeras reacciones, posteriormente se irán dando pequeñas indicaciones a los usuarios sobre el manejo de la aplicación. La primera interacción en probar fue la interacción usando las dos manos y la siguiente fue la interacción usando las órdenes por voz.

Para leer sobre los resultados y conclusiones sacadas de la prueba leer en el Anexo 1 los puntos 3.2 y 3.3. Este anexo recoge el artículo *"Home videogame devices like Kinect as educational tools for children with special needs"* este artículo ha sido presentado al congreso IDC 2012. Una parte de este artículo (punto 3) habla del sistema PonerMesa y de las conclusiones que se pueden sacar tras ver las reacciones de los niños ante los dos tipos de interacción.

5.4 La aplicación CrearMesa, desarrollo

Como se ha explicado en el punto 4.3, la aplicación PonerMesa lee los huecos de la mesa de un XML. La aplicación CrearMesa es una interfaz gráfica donde el usuario seleccionará la disposición deseada de los elementos de la mesa y la aplicación generará automáticamente el XML que será usado por PonerMesa. Para la interacción con esta aplicación no se usará Kinect sino que se utilizará el ratón.

Como se puede ver la figura siguiente la interfaz gráfica es similar a la de la aplicación PonerMesa, se tienen un conjunto de utensilios en la parte superior que son los utensilios sobre los que se puede hacer click sobre ellos y se podrán arrastrar hasta el lugar donde se quieran colocar. También será necesario colocar un mantel por cada comensal de la mesa. A la hora de generar el XML la aplicación sólo tendrá en cuenta los utensilios que estén sobre un mantel.

Figura 5.42 Interfaz gráfica aplicación CrearMesa

Otra diferencia gráfica con la aplicación PonerMesa es que en esta aplicación para el utensilio seleccionado (este o no en desplazamiento) aparecerá a la derecha unos paneles con opciones:

- Un primer panel, donde se podrán elegir los tipos de utensilio que se quiere que acepte el hueco que va a ocupar el utensilio.

- Un segundo panel donde se seleccione la orientación del Utensilio.
- Un botón borrar por si se quiere borrar el utensilio.

Una vez se hayan colocado todos los utensilios se pulsará el botón generate, que creará el fichero XML que describa la disposición de los objetos elegida por el usuario. Una vez creado el archivo se validará con el esquema XSD correspondiente, si se valida se informará al usuario de que se ha generado el fichero correctamente, si no se informará del error y se borrará el fichero.

5.4.1 La clase Boton.cs y herencia

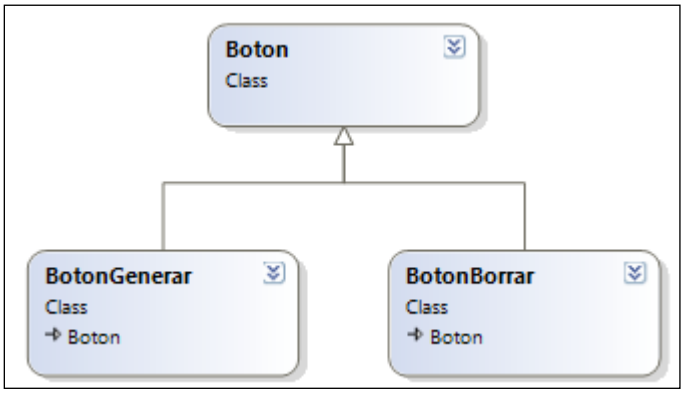


Figura 5.43 Herencia de Boton.cs

Como se puede ver en la imagen se tienen dos botones, el botón generar y el borrar, su funcionamiento es similar, pero varía en algunas cosas, por lo tanto se ha decidido crear una clase general llamada Boton.cs, que recogiera el comportamiento general y dos clases hijas llamadas BotonBorrar.cs y BotonGenerar.cs.

Las variables de esta clase son las siguientes:

- Width y Heigth son dos constantes que determinan el tamaño del botón.
- iconoBoton es la imagen del botón.
- letraTitulo es el spriteFont que se usará para escribir el texto de dentro del botón.
- Posición y rectanguloBoton guardarán la posición y el rectángulo (sprite) del botón.
- La variable hayQueMostrar es una variable Booleana que indicará si es necesario dibujar el botón o no.
- estaHaciendoClick es una variable que dice si se estaba haciendo click en el instante anterior.

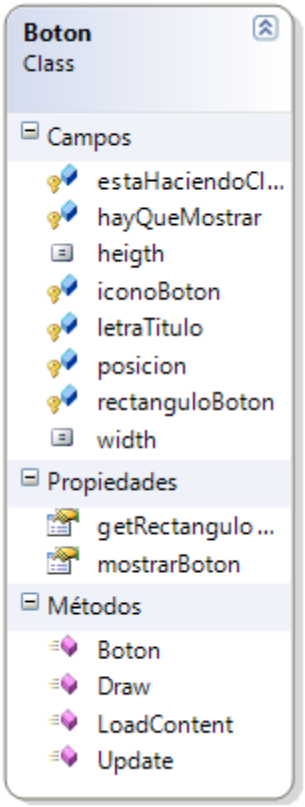


Figura 5.44 Esquema de la clase Boton.cs

Los métodos de la clase son:

- Boton, es el constructor, recibe como parámetro la posición y a partir de esta y las constantes inicia la variable rectanguloBoton.
- LoadContent, aquí se carga tanto la imagen del botón como el spriteFont.
- Update, calcula si el ratón esta encima del botón y está haciendo click, en ese caso pone el valor de estaHaciendoClick a true, en caso contrario lo pone en false.
- Draw, si la variable hayQueMostrar es verdadera dibujará el botón.

Las propiedades que son accesibles desde el exterior permitirán elegir si se muestra o no el botón (mostrarBoton) o devolverán el rectángulo del botón (getRectanguloBoton).

La clase BotonBorrar.cs además extenderá la funcionalidad de Boton haciendo que cuando se haga click se emita un evento nuevo, suprimirUtensilio, que hará que el objeto asociado a este Boton sea borrado.

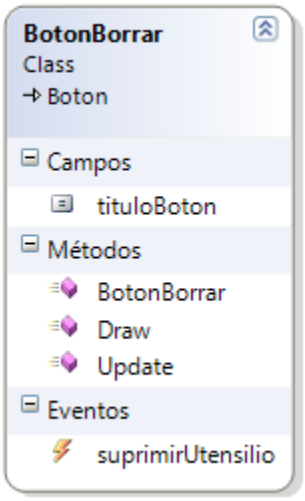


Figura 5.45 Esquema Clase BotonBorrar

Las dos diferencias con su clase padre son:

- En el método Update al detectar que se hace click, es decir, cuando el mouse esta sobre el botón haciendo click y la variable estaHaciendoClick es falsa, lanza el evento suprimirUtensilio.
- En el método Draw dibuja el texto contenido en la constante tituloBoton (en el centro del botón).

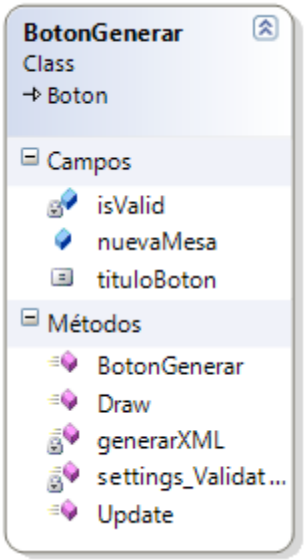


Figura 5.46 Esquema clase BotonGenerar

La diferencia con la clase padre es:

- En el método Update cuando se hace click se llama al método generarXML.
    - El método generar XML creará el XML a partir de la lista de manteles de la clase NuevaMesa.cs, esta lista está formada por los manteles que ha colocado el usuario junto con información de los utensilios que hay en él. Una vez generado el XML será validado por el esquema XSD en caso de

que sea favorable se mostrará al usuario un mensaje, en caso contrario se informará al usuario y se borrará el archivo XML.

- La variable isValid dice si el XML generado es correcto o no.
- EL método settings_ValidationEventHandler es un método que salta durante el proceso de validación si se produce algún error, si salta pone la variable isValid a false.
- El método Draw, al igual que en BotonBorrar dibujará el botón cuando la variable hayQueMostrar es true, dibujando el String contenido en la constante tituloBoton.

5.4.2 La clase BotonCheck.cs y herencia

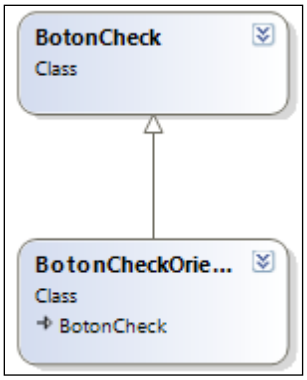


Figura 5.47 Herencia de BotonCheck

En los cuadros que aparecen en la parte derecha de la pantalla están las diferentes opciones a elegir para el utensilio seleccionado (objetoSeleccionado), estas diferentes opciones vienen determinadas por checkBox, estos checkBox es necesario implementarlos ya que no vienen en XNA. El motivo de la implementación de la clase BotonCheckOrientación es que se quiere añadir la funcionalidad de que cuando se cambie el estado de un checkBox "avise" al resto, por ejemplo lanzando un evento. Esto es útil en la elección del tipo de orientación, ya que al poder elegir un tipo de orientación es conveniente que al seleccionar un tipo se deseleccionen los demás.

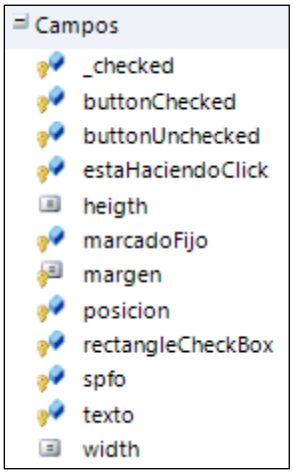


Figura 5.48 Campos de la clase BotonCheck

Entre los campos se puede destacar:

- Width y Heigth son dos constantes que determinan el tamaño del CheckBox.
- Margen es una constante que indica la distancia que debe haber entre el checkBox y su texto.
- _checked es una variable Booleana que indica si esta chequeado o no el checkBox.
- buttonChecked y buttonUnchecked son las dos imágenes del checkBox, chequeado y sin chequear respectivamente.
- estaHaciendoClick, sirve para saber si el ratón al estar pulsado es un nuevo click o es que ya estaba anteriormente pulsado.
- marcadoFijo, esta variable si es true indicará que ese checkBox estará siempre chequeado.
- Posición y rectangleCheckBox guardan la posición y el rectángulo que ocupa la imagen del checkBox.
- Texto, es el texto que se mostrará a la derecha del botón.
- Spfo es el spriteFont utilizado para dibujar el texto que va a la derecha del botón.

Los métodos de la clase son los propios de XNA y son llamados por los métodos análogos de la clase desde la que se llama a esta clase.

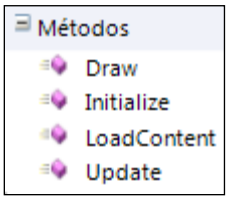


Figura 5.49 Métodos de la clase BotonCheck

- El método Initialize recibe como parámetros de entrada la posición del checkBox, el texto asociado y un valor booleano que será guardado en la variable marcadoFijo.
- LoadContent, se cargará tanto el spriteFont como las imágenes del checkBox seleccionado y sin seleccionar.
- Update, comprobará si se ha hecho clic encima del checkBox, es decir si el ratón está pulsado encima del botón y la variable estaHaciendoClick es igual a false, en ese caso se cambiará de valor la propiedad _checked (de true a false i viceversa).
- Draw, dibujará una imagen u otra en función del valor de _checked, además dibujará a su derecha el texto que se introdujo al método Initialize.

En la clase BotonCheckOrientación lo único que se sobrescribirá será el método Update, haciendo que cada vez que se haga click sobre un checkBox lo ponga como seleccionado y lance un evento (cambioCheck) con la intención de advertir que ha cambiado de estado (en la clase CuadroOpciones se tratará esta excepción poniendo el resto de checkBox como no seleccionados).

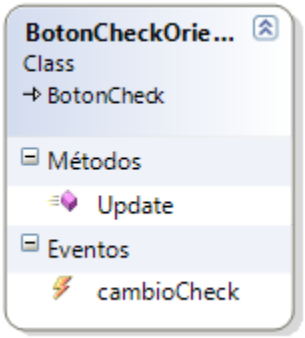


Figura 5.50 Esquema de la clase CheckBoxOrientacion

5.4.3 La clase CuadroOpciones.cs

Como se ha dicho anteriormente al tener un utensilio seleccionado aparecerá a la derecha unos paneles donde aparecerán distintas propiedades, la clase CuadroOpciones es quien dibuja estos paneles. La forma de estos paneles es un rectángulo exterior negro y dentro un rectángulo interior blanco que es quien contendrá las diferentes opciones a elegir (objetos de la clase BotonCheck).

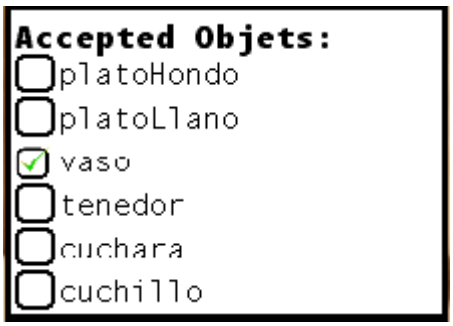


Figura 5.51 Ejemplo de dibujo del Objeto CuadroOpciones

Existen distintos tipos de CuadroOpciones, se ha creado una enumeración que enumera los diferentes tipos de Cuadros, por ahora hay dos tipos tipoUtensilio y tiposOrientacion.

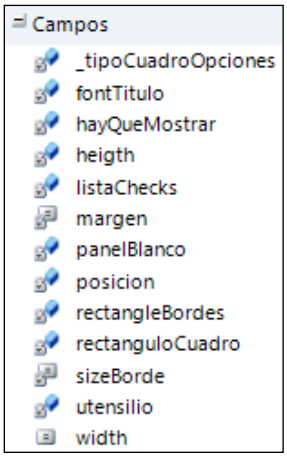


Figura 5.52 Campos de la clase CuadroOpciones

Los campos de la clase son los siguientes:

- _tipoCuadroOpciones, dice que tipo de CuadroOpciones es.
- fontTitulo, es el spriteFont del título del cuadro.
- hayQueMostrar, variable booleana que indica si hay que dibujar el cuadro o no.
- Width es una constante que dice el ancho del cuadro exterior.
- sizeBorde, es la cantidad de pixeles que sobresale el cuadro exterior del interior.
- Heigth es una variable que depende del número de elemento BotonCheck que contiene el cuadro.
- listaChecks, es una lista que guarda todos los objetos de la clase BotonCheck que contiene el cuadro.
- Margen, es la separación que debe haber entre los diferentes objetos BotonCheck.
- panelBlanco, es la imagen del cuadro, es un rectángulo blanco.
- Posición es la posición que ocupa el rectángulo exterior.
- rectangleBordes, rectanguloCuadros son los rectángulos de los cuadros exterior e interior respectivamente.
- Utensilio, es una referencia al utensilio asociado a este cuadro.

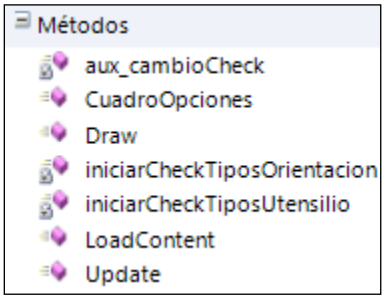


Figura 5.53 Métodos de la clase CuadroOpciones

Los métodos son los siguientes:

- CuadroOpciones, es el constructor, recibe como parámetros de entrada la posición del cuadro, el tipo de cuadro y una referencia al utensilio asociado al cuadro. En el constructor carga también los diferentes botones BotonCheck que contendrá el cuadro, para ello llamará a un método u otro en función del tipo de CuadroOpciones con el que se esté trabajando.
    - Si es un cuadro del tipo tiposUtensilio, se llamará al método iniciarCheckTiposUtensilio, en este método se hará un recorrido por cada elemento del diccionario getUtensilios de la clase Utensilio (ahí están todos los tipos de utensilio) instanciando un objeto BotonCheck para cada elemento. Al elemento cuyo tipo coincida con el tipo de utensilio asociado se le asignará valor true al parámetro del constructor que hace referencia a la variable “marcadoFijo”,
    - Si el cuadro es del tipo tiposOrientacion se usará el método iniciarCheckTiposOrientacion, en este al igual que en el método anterior se creará un objeto BotonCheck para cada orientación posible, poniendo por defecto seleccionada la orientación “arriba”.
- auxCambioCheck, este método es el que recoge el evento cambioCheck lanzado por los objetos de la clase BotonCheckOrientacion en los cuadros de tipo tiposOrientacion. Este método recorre

todos los BotonCheck de la lista listaChecks poniéndolos como no seleccionados, a excepción del que lanzo la excepción que aparece como seleccionado.

- En el método LoadContent, se cargará el spriteFont del título y la imagen del cuadro, además se llamará al método LoadContent de cada elemento BotonCheck contenido en el cuadro.
- El método Update llamará a los métodos Update de todos los elementos BotonCheck del cuadro.
- En el método Draw, si la variable hayQueMostrar es true, dibujará lo rectángulos exterior e interior, pintará el texto del título y llamará a los métodos Draw de cada BotonCheck contenido en listaChecks.

Las propiedades accesibles desde el exterior se pueden ver en la siguiente figura:

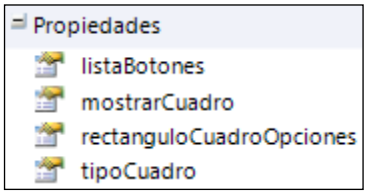


Figura 5.54 Propiedades de la clase CuadroOpciones

La propiedad rectanguloCuadroOpciones devolverá el rectángulo exterior del cuadro.

5.4.4 La clase Utensilio y herencia

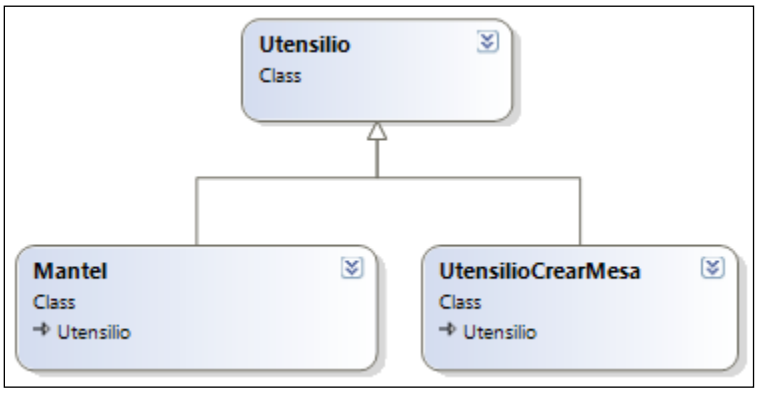


Figura 5.55 Herencia de la clase Utensilio

Debido al cambio de aplicación ha sido necesario cambiar algo el comportamiento de la clase Utensilio utilizada en la aplicación PonerMesa. Es por eso que se han creado las clases Mantel y UtensilioCrearMesa que heredan de la clase Utensilio y refinan su comportamiento adaptándolas a la nueva aplicación.

La clase UtensilioCrearMesa gestionará los utensilios que se cogen de la parte superior de la pantalla, es necesario crear una clase heredada ya que se quieren añadir las siguientes funcionalidades.

- Será capaz de cambiar de orientación si el usuario cambia de orientación en el CuadroOpciones.
- Será capaz de saber si esta dentro de un elemento Mantel o no.

Los campos que tiene la nueva clase son:

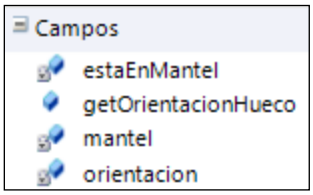


Figura 5.56 Campos de la clase UtensilioCrearMesa

El campo estaEnMantel es un booleano que indica si el utensilio se encuentra sobre un mantel, en ese caso la variable mantel contendrá una referencia a ese mantel.

El campo orientación guarda la orientación del utensilio en ese momento, cuando en el CuadroOpciones se cambia de orientación llama a la propiedad OrientacionUtensilio para cambiar la orientación del utensilio y que se dibuje con la nueva orientación en tiempo real.

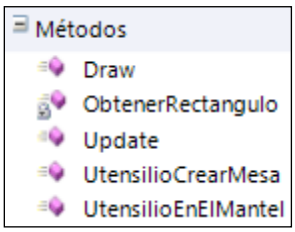


Figura 5.57 Métodos de la clase UtensilioCrearMesa

El método ObtenerRectángulo es el mismo que el utilizado en la clase HuecoUtensilio, da un rectángulo según la orientación para que al aplicar la rotación quede en el sitio correcto.

UtensilioEnElMantel calcula si el utensilio está dentro del mantel que se le pasa como parámetro, de ser así actualiza la variable mantel y pone la variable estaEnMantel a true.

El constructor UtensilioCrearMesa recibe los parámetros de entrada del constructor de la clase Utensilio más la orientación del utensilio.

El método Draw dibujará el utensilio aunque utilizando distintos colores según en el estado en que se encuentre:

- Si lo han dejado dentro de un mantel, color blanco.
- Si esta seleccionado y en movimiento, color rojo.
- Si esta seleccionado pero no en movimiento, color verde.

Además de la propiedad OrientacionUtensilio ya comentada existen las propiedades SeEncuentraEnElMantel, que permite acceder y modificar al valor del campo estaEnElMantel, y MantelContenedor, que permite obtener el mantel donde está el utensilio.

Para la clase Mantel se deseaban hacer las siguientes modificaciones de la clase Utensilio:

- Se trata de un Utensilio que contiene otros Utensilios, por lo tanto debe tener una lista con todos los Utensilios que contiene y otra con los tipos de utensilios que contiene.
- El mantel, gráficamente hablando, tiene dos estados, el icono que esta a mano izquierda de la pantalla esta en modo plegado, pero al hacer click sobre él se hace una copia de este pero en modo desplegado, que es más grande, que será el que se coloque en la mesa.
- Al mover el mantel es deseable que todos los utensilios que están colocados en él se desplacen con él.

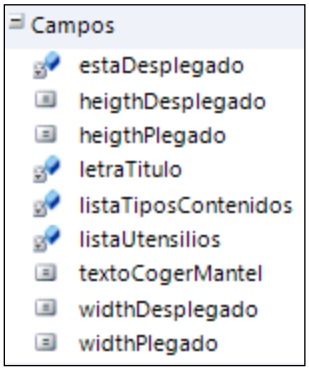


Figura 5.58 Campos de la clase Mantel

Los campos heigthDesplegado, heigthPlegado, widthDesplegado y widthPlegado son constantes que guardan el tamaño del mantel en sus dos estados posibles.

Cuando el mantel está plegado aparece encima de él un mensaje, este mensaje está guardado en la constante textoCogerMantel, para dibujar el texto se usará el spriteFont letraTítulo.

La variable Booleana estaDesplegado dirá si el Mantel se encuentra plegado o desplegado.

Las listas listaTiposContenidos y listaUtensilios son listas sobre los tipos contenidos en el mantel (sin repetición) y los utensilios que están colocados dentro de él respectivamente.

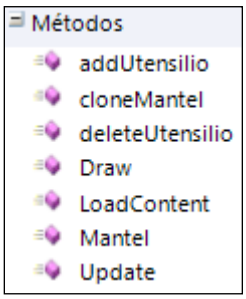


Figura 5.59 Métodos de la clase Mantel

El método cloneMantel es similar al cloneUtensilio, devolverá un objeto de la clase mantel copia exacta del objeto en el que se encuentra, es usado cada vez que se hace click en el mantel que se encuentra en la derecha de la pantalla para coger un nuevo mantel.

Cada vez que se quiere añadir un utensilio a un mantel se llama al método addUtensilio, este método añadirá el utensilio a la lista listaUtensilios y añadirá el tipo del utensilio a la lista listaTiposContenidos si este no se encontraba todavía en la lista.

Para eliminar el utensilio por el contrario se llamará al método deleteUtensilio, que además de borrar el utensilio de listaUtensilios, borrará el tipo de listaTiposContenidos si no hay más elementos del mismo tipo en listaUtensilios.

El constructor de la clase Mantel tendrá los mismos parámetros de entrada que la clase Utensilio más un valor booleano que indique si se encuentra desplegado o no, en el caso de estar desplegado se instanciarán también las listas, cosa que no se hará en caso de que el mantel esté plegado ya que no se podrán dejar utensilios en un mantel plegado.

El método Update calculará la variación que se ha producido entre la posición del ratón y el punto central del Mantel aplicando este mismo desplazamiento a todos los utensilios que se hallen colocados en el mantel.

En el método Draw se dibujará el mantel (tamaño del rectángulo diferente dependiendo de que este plegado o desplegado) y en el caso de que esté plegado se escribirá encima el contenido de la constante textoCogerMantel (cuyo SpriteFont letraTitulo había sido cargado en el método LoadContent).

5.4.5 La clase NuevaMesa.cs

Es la clase principal, hereda de Game.cs y es la que lleva el “peso” de la ejecución. Desde esta clase es desde donde se llamará al resto de clases a lo largo de la ejecución del programa.

Debido a que un Utensilio tiene asociados unos cuadros de propiedades y un botón de borrado se ha creado una estructura que asocie cada Utensilio a sus cuadros de propiedades y su botonBorrar, esta estructura se llama PropiedadesUtensilioColocado.

```
public struct PropiedadesUtensilioColocado
{
    public Utensilio objeto;
    public List<CuadroOpciones> listaCuadrosOpciones;
    public BotonBorrar botonBorrar;
}
```

Figura 5.60 Estructura PropiedadesUtensilioColocado

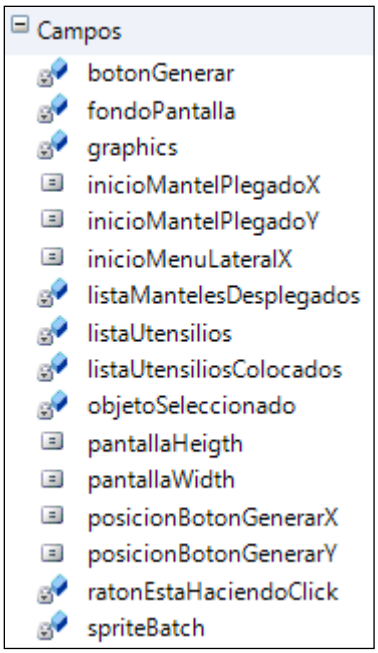


Figura 5.61 Campos de la clase NuevaMesa

Como se puede ver los campos de la clase (exceptuando los propios de XNA) son los siguientes:

- pantallaWidth y pantallaHeigth son las constantes que determinan el tamaño en pixeles del campo de juego.
- posicionBotonGenerarX y posicionBotonGenerarY son las constantes que guardan la posición del objeto botonGenerar.
- inicioMantelPlegadoX e inicioMantelPlegadoY, son dos constantes que guardan la coordenada donde se dibujará el mantel plegado (aquel sobre el que hay que pulsar para obtener los manteles de cada comensal).
- inicioMenuLateralX, coordenada X donde se dibujarán los paneles laterales y el botonBorrar, las diferentes coordenadas Y dependerán del tamaño de los cuadrosOpciones anteriores y del tamaño del mantel plegado.
- fondoPantalla es la imagen que se mostrará como fondo de pantalla.
- listaUtensilios, al igual que en la clase Game1 guardará la lista de Utensilios disponibles que se encuentran en la parte superior de la pantalla, a todos estos además se le añadirá el objeto de la clase Mantel que se encuentra a la derecha de la pantalla.
- listaMantelesDesplegados, es una lista de objetos de la estructura PropiedadesUtensilioColocado que contiene a todos los objetos Manteles que están desplegados encima de la mesa (cada mantel a su vez contiene todos los utensilios que están colocados en el).
- listaUtensiliosColocados, es otra lista de objetos PropiedadesUtensilioColocado que contiene referencias a todos los objetos Utensilio (a excepción de los de la clase Mantel) que están encima de la mesa, estén o no colocados en un mantel.
- objetoSeleccionado, es el objeto de la lista listaUtensiliosColocados que se haya seleccionado en ese momento (puede estar moviéndose o no), este objeto será del que se muestren los cuadros de opciones y botón de borrado a la derecha de la pantalla.

- ratonEstaHaciendoClick, es un valor booleano que indica si en el instante anterior el ratón estaba pulsado o no.
- botonGenerar, es el objeto de la clase BotonGenerar que representa al botón de la parte de debajo de la pantalla.

Los métodos implementados en esta clase son los siguientes.

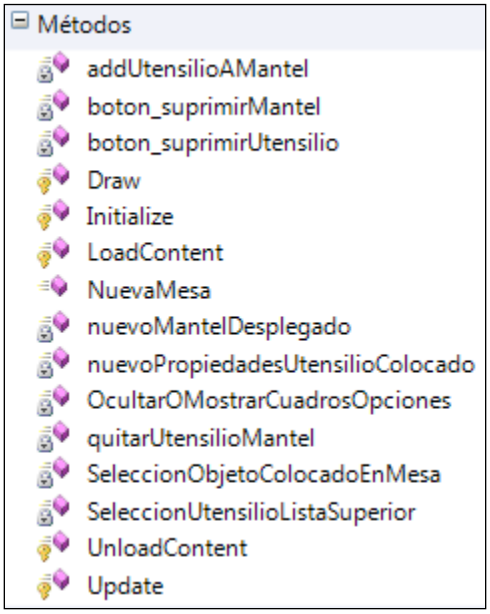


Figura 5.62 Métodos de la clase NuevaMesa

- addUtensilioMantel, este objeto es llamado cuando un utensilio ha dejado de moverse, devuelve true si el utensilio ha sido dejado encima de un mantel, para ello se recorren todos los manteles de la lista listaMantelesDesplegados llamados al método UtensilioEnElMantel de estos, si devuelve true se llamará al método addUtensilio de la clase Mantel y se devolverá true.
- quitarUtensilioMantel, se hace que el utensilio deje de estar ligado al mantel donde se encontraba llamando al su propiedad SeEncuentraEnElMantel para ponerla a false y llamando al método deleteUtensilio de la clase Mantel.
- SeleccionObjetoColocadoEnMesa, este método devolverá true si se ha hecho click sobre algún objeto de la lista que se le pasa como parámetro de entrada. En el caso de que la posición del mouse (pasada como parámetro de entrada) este sobre uno de los objetos se procederá a:
    - Llamar al método OcultarOMostrarCuadrosOpciones para que muestre los cuadros de opciones del utensilio.
    - Se llamará a su propiedad estaEnDesplazamiento para darle el valor true.
    - Si el utensilio estaba colocado en un mantel se llama al método quitarUtensilioMantel para sacarlo del mantel.
    - Se toma este objeto como nuevo objetoSeleccionado.
- OcultarOMostrarCuadrosOpciones, se encargará de mostrar u ocultar (dependiendo del valor booleano pasado en los parámetros) los cuadros de opciones y el botón de borrado del objeto

PropiedadesUtensilioColocado pasado como parámetro. Hará un recorrido por los diferentes objetos CuadroOpciones llamando a su propiedad mostrarCuadro dándole el valor del booleano pasado como parámetro, se hará lo mismo con el objeto botonBorrar.

- SeleccionUtensilioListaSuperior, para cada objeto del tipo PropiedadesUtensilioColocado de la lista listaUtensilios se comprobará si el ratón esta encima de ellos (valor de la posición del ratón pasado como parámetro). De ser así este objeto será el nuevo objetoSeleccionado y se pasará a su propiedad estaEnDesplazamiento el valor true para indicar que está en movimiento. Según el tipo de clase del objeto Utensilio de la estructura se llamará a diferentes métodos:
    - Si es del tipo Mantel, se llamará al método nuevoMantelDesplegado, el objeto que devuelva será añadido a la lista listaMantelesDesplegados.
    - Si es del tipo UtensilioCrearMesa, se llamará al método nuevoPropiedadesUtensilioColocado y se añadirá el objeto devuelto a la lista listaUtensiliosColocados.
- NuevoMantelDesplegado, recibiendo como variable de entrada un objeto de la clase Mantel, devolverá el objeto del tipo PropiedadesUtensilioColocado asociado. Para ello se instanciarán su objetos CuadroOpciones correspondientes (que serán añadidos al campo listaCuadrosOpciones) y su botonBorrar al que se indicará como método que tratará el evento suprimirUtensilio el método botón_suprimirMantel().
- NuevoPropiedadesUtensilioColocado, devolverá a partir de un objeto del tipo UtensilioCrearMesa un objeto del tipo PropiedadesUtensilioColocado que lo contenga. Instanciará sus objetos CuadroOpciones correspondientes (que serán añadidos al campo listaCuadrosOpciones) y su botonBorrar al que se indicará como método que tratará el evento suprimirUtensilio el método botón_suprimirUtensilio().
- Botón_suprimirUtensilio, método que realiza las acciones asociadas al evento suprimirUtensilio, se borrará el objeto objetoSeleccionado(es el único que se puede hacer click en su botón borrar) de la lista listaUtensiliosColocados y se igualarán a null todos sus campos, finalmente la variable objetoSeleccionado se pondrá a null también.
- Boton_suprimirMantel, cuando el objetoSeleccionado es del tipo Mantel y se da a su botón de borrar se ejecutará este método. Se recorrerán todos los utensilios que contiene pasando a su propiedad SeEncuentraEnElMantel el valor false (para indicar que ya no se encuentran colocados en un mantel). Una vez hecho esto todos los campos del objeto se pondrán a null para después ser la propia variable objetoSeleccionado la que se iguale a null.

- NuevaMesa, constructor de la clase.

- Initialize, se crean la tres listas, la listaUtensilios es rellenada usando la clase LectorXML y se crea un objeto Mantel (plegado) que es añadido a esta lista. También se crea el objeto BotonGenerar.

- LoadContent, se carga la imagen de fondo de la pantalla y se llama a los métodos LoadContent del objeto de la clase BotonGenerar y al de todos los objetos de la lista listaUtensilios.

- En la clase Draw se pintará el fondo de la pantalla y después se llamará a los métodos Draw de los objetos en el siguiente orden:

- Cada objeto de listaUtensilios.
- Cada objeto de listaMantelesDesplegados.
- Cada objeto de listaUtensiliosColocados.
- Si existe objetoSeleccionado, se llamará al método Draw de cada campo que compone la estructura (se dibujará el utensilio, sus cuadros de opciones y su botón de borrado).
- El del objeto de la clase BotonGenerar.

- El método Update es el que maneja la dinámica del juego, según el estado del ratón y la variable ratonEstaHaciendoClick se manejarán las posibles situaciones que se pueden dar:
  - Si esta el ratón presionado y ratonEstaHaciendoClick es falso, quiere decir que el usuario acaba de presionar el ratón. En este caso se pondrá la variable a true y se llamará a los siguientes métodos en este orden:
    - SeleccionObjetoColocadoEnMesa pasando como parámetro la lista listaUtensiliosColocados.
    - En el caso que devuelva false, se llama al método anterior pero esta vez pasado la lista listaMantelesDesplegados.
    - Si el método anterior devuelve false se llamará al método SeleccionUtensilioListaSuperior.
  - Si el ratón está presionado y ratonEstaHaciendoClick es verdadero, quiere decir que el mouse lleva un rato pulsado. En este caso se comprobará que exista el objetoSeleccionado y que este se halle en movimiento, si es así, se llamará a su método Update para que actualice su posición con la del ratón.
  - Si el ratón no está presionado, se comprobará si la variable ratonEstaHaciendoClick es true y que el objetoSeleccionado existe (es distinto de null), ese caso quiere decir que es posible que se haya dejado de mover el objeto, por lo que se pondrá su propiedad estaEnDesplazamiento a false y si el objeto contenido en el campo utensilio es del tipo UtensilioCrearMesa se llamará a su método addUtensilioMantel para comprobar si ha sido colocado en un mantel. La variable ratonEstaHaciendoClick se pondrá como valor false.

Una vez hecho esto, si el objetoSeleccionado es distinto de null se llamará a los métodos Update de sus objetos CuadroOpciones y BotonBorrar. También se llamará al método Update de botonGenerar.

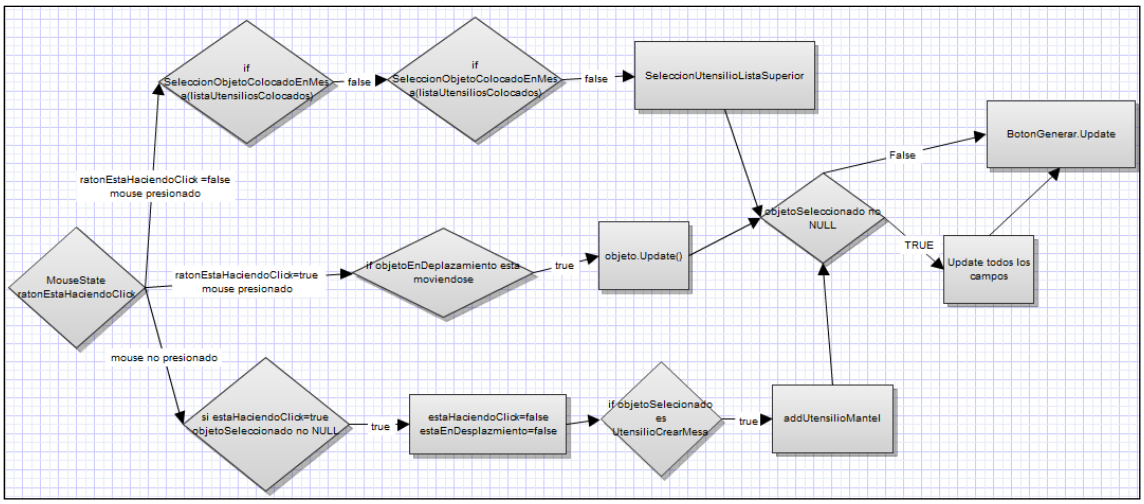


Figura 5.63 Diagrama de flujo de NuevaMesa.Update

Capítulo 6 Kinect aplicado en el mundo del arte

El sistema Kinect tiene muchas posibilidades, por ejemplo el mundo del arte. Isabel Sánchez Gil, licenciada en Bellas artes, va a realizar una exposición de sus obras en la ciudadela a partir del próximo 27 de abril. Como complemento a esta exposición deseaba añadir una pequeña aplicación de Kinect donde el usuario pudiera interactuar con una aplicación donde aparecieran sus obras, la intención final es que el usuario pudiera mostrar su propia visión de la realidad.

Esta aplicación deberá realizar lo siguiente:

- Nada más detectar al usuario la pantalla deberá cambiar de color con el objetivo de captar la atención del usuario. Además deberá mostrar la sombra del usuario con el objetivo de animar a este a que se mueva.
- Cuando el usuario estire uno de los dos brazos aparecerá en este una figura perteneciente a una de las obras de Isabel, en el otro brazo no podrá aparecer otra obra hasta que el usuario haya dejado esta. La figura rotará en función de la pendiente que se forma entre el centro de la imagen y la mano desocupada.
- Cuando el usuario coge una figura la sombra de este se irá desvaneciendo de la pantalla.
- Para dejar una figura bastará con que el usuario cruce los brazos.
- Si el usuario esta un rato sin coger ninguna figura su sombra volverá a aparecer y desaparecerá cuando este coja una nueva figura.
- Cuando el usuario salga de la pantalla el sistema buscará otro nuevo usuario, cuando lo encuentre borrará la composición del usuario anterior y pintará la pantalla de un nuevo color.

Figura 6.1 Ejemplo interfaz aplicación

Es importante recalcar que con lo que va a trabajar el usuario es con pinturas de la artista o con fragmentos de estas, se desea que en cada usuario solamente pueda trabajar cada vez con una pintura y sus recortes, no siendo posible dentro de una misma partida mezclar elementos de distintas obras, es decir podemos separar los dibujos en "familias de dibujos".

Como se puede ver en el diagrama de clases, las clases que forman el sistema son las siguientes:

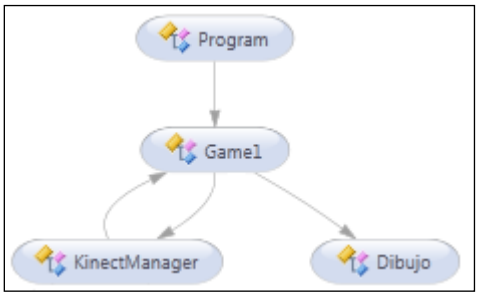


Figura 6.2 Diagrama de clases

- Program.cs, clase de XNA, es la que llama a la clase Game1.
- Game1.cs,
- KinectManager, es la encargada de todo lo relacionado con el reconocimiento del jugador, detectará la posición de las manos y si el brazo está estirado o no, también se encargará del dibujo de la sombra en la pantalla.
- Dibujo, esta clase se encargará de cargar y dibujar los diferentes dibujos (pertenecientes a la colección de Isabel) que se mostrarán por pantalla.

6.1 La clase KinectManager.cs

Como se ha dicho anteriormente esta clase es la encargada de detectar el movimiento del jugador y de decir cuando se cumplen las condiciones para que se dibuje un objeto de la clase Dibujo y cuando hay de dejarlo de mover. Además es la clase que se encargará de dibujar la sombra del jugador.

Para dibujar la sombra del jugador se recurrirá a la cámara de profundidad, dibujando aquellos pixeles cuyo ID sea el mismo que el del esqueleto que se está leyendo. Cuando se detecte que se debe dibujar un dibujo la sombra se irá desvaneciendo paulatinamente. Si después de dejar un objeto el usuario está inactivo la sombra volverá a aparecer.

Para el desarrollo de la aplicación ha sido necesario crear una serie de estructuras:

- ConjuntoCoordenadasMano, guarda las coordenadas convertidas al espacio de juego y las originales del skeleton.
- MostrarOcultarFade, formado por dos variables booleanas que indicarán si hay que ir ocultando o mostrando la sombra paulatinamente.

```
struct MostrarOcultarFade
{
    public Boolean mostrar;
    public Boolean ocultar;
}
```

Figura 6.3 Estructura MostrarOcultarFade

- LadoEstirado, es necesario saber qué lado es el que el jugador ha estirado y en ese momento es el que maneja el dibujo que está en movimiento, esta enumeración guarda las diferentes opciones.

```
public enum LadoEstirado
{
    derecho,
    izquierdo,
    none
}
```

Figura 6.4 Enumeración LadoEstirado

Uno de los requisitos era que al reconocer usuario se cambiara el fondo de la aplicación siguiendo una determinada secuencia, para conseguirlo lo que se hace es ligar cada color a un numero por medio de un diccionario de manera que a partir de una variable contador que vaya ascendiendo (y que se ponga a cero cuando sea igual al número de elementos en el diccionario) se podrá acceder secuencialmente a los diferentes colores de la pantalla. El diccionario se llama getColorPantalla.

La clase también contendrá una serie de constantes que sirven como parámetros que regulan el funcionamiento del programa:

- NumeroMilisegundosMaximosSinJugador, es el tiempo que esperará la aplicación sin detectar nuevos esqueletos del esqueleto que está analizando. Una vez pase este tiempo cambiará el esqueleto en detección y cambiará de jugador, borrando la composición del antiguo, cambiando el color de fondo y redibujando la sombra del nuevo jugador.
- elevationCamera, es la inclinación que toma el dispositivo Kinect.
- milisegundosEfectoFade, es el tiempo que le cuesta aparecer o desparecer.
- segundosSinInteractuar, es el tiempo que se espera sin que el jugador estire el brazo (es decir que haya que dibujar un nuevo dibujo) para volver a dibujar la sombra del usuario.
- colorSilueta, es el color con el que dibujará la sombra del jugador.

La clase también lanzará diferentes eventos cuando se produzcan diferentes acontecimientos:

- CogerDibujo, al estirar el brazo se lanzará este evento.
- DejarDibujo, será lanzado al estirar el brazo.
- NuevoJugador, cuando se detecte un nuevo Jugador este evento será lanzado.

El resto de campos que definen la aplicación son:

- skeletonTrackingIdJugador, guarda el tracking id del jugado que se está reconociendo en ese momento.
- idDepthADibujar, es el id del jugador que aparece en los pixeles de la cámara de profundidad que pertenecen al jugador que se quiere dibujar su sombra. Este id no se corresponde al trackingId del esqueleto sino a la posición de ese skeleton dentro del array de skeletons.
- ultimaLecturaEsqueletoJugador, es un campo del tipo DateTime que guarda la fecha de la última vez que se trato el skeleton del jugador.
- _brazoEstirado, guarda el valor del brazo que está controlando un objeto de la clase dibujo.
- kinectDepth, es un objeto del tipo Texture2D donde se dibujará la sombra del jugador.
- ultimoEventoDejar, con el fin de evitar problemas en la ejecución se decide guardar la fecha del último evento dejar para que después de un evento DejarDibujo tenga que pasar al menos un segundo antes de poder lanzar un evento CogerDibujo.
- tiempoSinEstirarBrazo.
- mAlphaValue, guardará el grado de transparencia de los pixeles de la sombra a dibujar, siendo 255 opacidad completa y siendo 0 la transparencia total.
- Width y Heigth guardarán el tamaño (en pixeles) de la pantalla.
- numeroColor, variable de tipo int, que servirá de contador, irá aumentando en uno hasta llegar al número que coincida con el número de elementos del diccionario, llegado a este punto será reiniciada a cero. Su función será servir de índice para acceder al diccionario de colores y así poder acceder a los colores de manera secuencial en el orden deseado.

Los métodos de la clase son los siguientes:

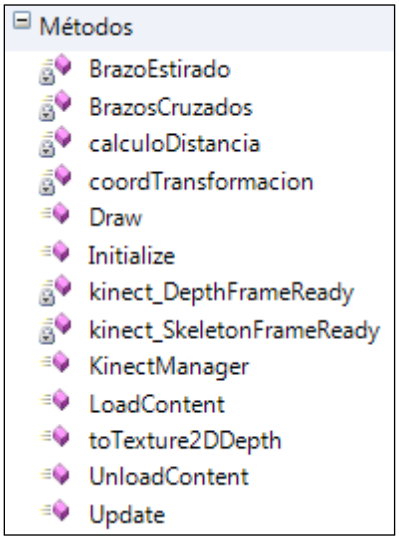


Figura 6.5 Métodos de KinectManager

- El método brazo estirado, este método lanzará el evento CogerDibujo, su hilo de ejecución es el siguiente:
    - Si no hay ningún brazo estirado y ha pasado más de un segundo después de la última vez que el usuario dejó un dibujo.

- Se llamará al método calculoDistancia, método que calcula la distancia entre dos Joints, si la distancia entre la mano y el hombro es similar a la distancia del brazo se considerara que el usuario ha estirado el brazo, en ese caso:
    - Se modificará la variable _brazoEstirado para que guarde el lado del brazo que acaba de estirar.
    - Se modificará cada campo de la estructura _MostrarOcularFade, para indicar que la sombra tiene que ir despareciendo
    - Se lanzará el evento CogerDibujo.

- El método brazos cruzados será el encargado de comprobar si las manos se han cruzado, acción que desencadenaría que se lanzara el evento DejarDibujo, el método comprobará que la variable _brazoEstirado no sea none (es decir, que hay un brazo manejando un dibujo) y si la mano derecha está más a la izquierda que la mano derecha hará lo siguiente:
    - Pone los DateTime ultimoElementoDejar y tiempoSinEstirarElBrazo con la fecha y hora de ese momento.
    - Se pone la variable _brazoEstirado como none.
    - Se lanza el evento DejarDibujo.
- El método kinect_skeletonFrameReady es el método que recoge el evento lanzado al crearse un nuevo frame con esqueletos. Se usará el mismo sistema que la aplicación PonerMesa para evitar que otros usuarios que estén en el terreno de juego interfieran. Sin embargo en este método se añadirán algunas cosas cada vez que se detecta un nuevo jugador:
    - Con la ayuda del contador numeroColor que se irá incrementando cada vez que aparezca un nuevo jugador y usándolo como índice para acceder al diccionario se podrá tener un color distinto al que se está mostrando en ese momento en la pantalla, este nuevo color será el nuevo color de fondo.
    - Se pondrá la variable mAlphaValue a 255 para que se muestre la sombra del nuevo jugador.
    - Se actualizará la variable skeletonTrackingIdJugador que guardará el trackingId del nuevo esqueleto a seguir.
    - La posición del nuevo esqueleto a seguir se guardará en la variable idDepthADibujar, ya que esta posición se corresponde con el id marcado en los pixeles que se corresponden al usuario en la imagen captada por la cámara de profundidad.
    - Se lanzará el evento NuevoJugador.

  Dentro del tratamiento del skeleton se hará lo siguiente:

    - Se calcularán las coordenadas adaptadas al tamaño de la pantalla del juego de los Joints de las dos manos (para ello se llamará al método coordTransformacion).
    - Si la variable _brazoEstirado es none se llamará al método BrazoEstirado (una vez para cada mano).
    - Si es igual a none, se llamará al método BrazosCruzados.
- El método kinect_depthFrameReady es el método que se ejecuta cada vez que se genera un nuevo Frame con datos de profundidad, este método llamará al método toTexture2DDepth que devolverá un texture2D a partir de la PlanarImage que pasa el evento.

- ToTexture2DDepth, este método recorrerá la PlanarImage y pintará en un objeto texture2D del mismo tamaño, solamente los pixeles que se corresponden con el jugador con el mismo índice que el guardado en la variable idDepthADibujar.
- KinectManager, es el constructor de la clase, recibe como parámetros el tamaño de la pantalla (Heigth y Width) y una referencia a la clase Game1. En el constructor también se elegirá el color de fondo que se use en la pantalla.
- Initialize, se iniciará Kinect, se abrirá el flujo de datos de profundidad así como se asociarán los métodos antes descritos a sus eventos correspondientes.
- LoadContent, se llamará al constructor de la clase Texture2D para instanciar el objeto kinectDepth.
- Update, se comprueban varias cosas:
    - Si el brazo no está estirado y ha pasado un tiempo mayor sin coger ningún objeto que el recogido en la constante tiempoSinEstirarBrazo, se modificará la variable _mostrarOcultarFade para que muestre paulatinamente la sombra del jugador.
    - Si esta en modo de ir mostrando paulatinamente la sombra se calculará el tiempo pasado desde el ultimo Update (game.elapsedGameTime()) y se calculará la proporción (respecto de la variable milisegundosEfectoFade) que hay que sumar a mAlphaFade.
    - En el caso de que haya que ir borrando la sombra se hará lo mismo pero esta vez restando la cantidad a mAlphaFade.
- Draw, dibujará el objeto kinectDepth que contiene la sombra del jugador.

Algunos campos es necesario que sean accedidos desde el exterior, para ello se han creado las siguientes Propiedades que permitirán acceder a las coordenadas adaptadas al campo de juego de la mano derecha e izquierda así como saber que brazo es el que tiene el control de un Dibujo.

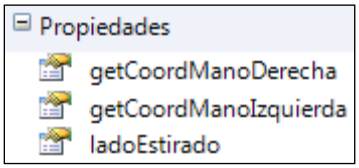


Figura 6.6 Propiedades clase KinectManager

6.2 La clase Dibujo.cs

Esta clase es la que se encarga de dibujar cada uno de los Dibujos que aparecen en pantalla. Cada vez que se instancie esta clase elegirá aleatoriamente una imagen dentro de la familia de imágenes que se le pasa por el constructor, esta imagen será la que se dibujará, para ello en esta clase se asociará en un diccionario el String con el nombre de la familia de imágenes a una lista con los nombres de las imágenes de esta familia. Generando aleatoriamente un número, se obtendrá el nombre de la imagen a dibujar en la posición marcada por el número dentro de la lista de imágenes de la familia. Además existe otro diccionario que asocia el nombre de la familia de imágenes con la escala a la que hay que dibujarlas (la escala es del tipo Vector2, donde se guarda la escala para cada uno de los ejes de la imagen).

- getNombreImagen, es el diccionario que asocia la el nombre de la familia de imágenes con la lista de los nombres de las imágenes de esa familia.
- getMedida, es el diccionario que asocia el nombre de la familia de imágenes con un objeto Vector2 que representa la escala a la que hay que dibujar cada imagen de esa familia.

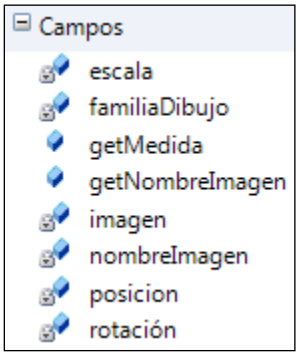


Figura 6.7 Campos de la clase Dibujo

Además de estos diccionarios, están los siguientes campos:

- rotación, este campo indica la rotación que hay que aplicar a la imagen (en radianes).
- Posición, guarda la posición de la imagen.
- nombreImagen, el nombre de la imagen.
- Imagen, Texture2D asociado al nombre de la imagen.
- Escala, objeto de la clase Vector2 que guarda la escala a la que hay que dibujar la imagen.
- familiaDibujo, String que guarda el nombre de la familia de dibujos que se está usando en ese momento.

Los métodos utilizados en la clase son los propios de XNA.

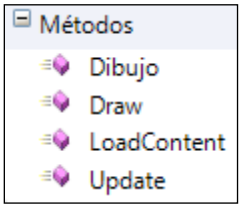


Figura 6.8 métodos de la clase Dibujo

- Dibujo, es el constructor, recibe como parámetro de entrada la posición donde debe dibujarse el dibujo (se corresponde con la posición de la mano del brazo que se ha estirado) y un String con la familia de dibujos con la que se está trabajando. Será en este método donde se genere el número aleatorio que dará el nombre de la imagen a mostrar a partir de la lista de imágenes (nombres de imágenes) asociada a la familia de imágenes (familiaDibujo) con la que se está trabajando.
- LoadContent, se cargará la imagen a dibujar.
- Update, se recibe un objeto de la clase Vector2 que actualizará la posición del dibujo.

- Draw, se dibuja la imagen aplicando la escala elegida y la rotación que se guarda en la variable rotación. Para conseguir que la imagen rote sobre su centro ha sido necesario prescindir del uso del objeto Rectangle a la hora de dibujar y se ha usado la variable Vector2 origin para indicar cual se quería que fuera el centro de la rotación.

La modificación de la variable rotación se hace desde el exterior de la clase (en el constructor se le asigna el valor cero), es por eso que se ha creado la propiedad modificacionRotacion que permite acceder al campo rotación y modificarlo.

6.3 La clase Game1.cs

Al igual que en otras aplicaciones, la clase Game1 es la que lleva el peso de la aplicación, es la que coordina el juego.

En la clase están definidas tres constantes, que definen el tamaño en pixeles de la ventana del juego y el nombre de la imagen que será utilizada como fondo (será una imagen en blanco), estas constantes son respectivamente:

- pantallaWidth
- pantallaHeigth
- textoImagenFondo

Además de estas constantes existe el diccionario getNombreFamiliaImagen, que ligará las familias de imágenes a números enteros. De esta manera generando un número entero aleatorio se podrá obtener de manera aleatoria la familia de imágenes a dibujar.

Además de los campos propios de XNA como graphics y spriteBatch, se tienen los siguientes campos:

- fondoAplicacion, elemento del tipo Texture2D asociado a la imagen con nombre contenido en la constante textoImagenFondo.
- colorPantalla, es el color con el que se pintará el fondo de la pantalla.
- _kinectManager, es una referencia a la clase KinectManager que realiza el tracking del esqueleto del jugador y dibuja su sombra.
- ladoConDibujo, es del tipo KinectManager.LadoEstirado, guarda el brazo que esta estirado, es decir, aquél que controla el movimiento de un objeto Dibujo.
- dibujoEnMovimiento, objeto de la clase Dibujo que en ese mismo instante está siendo controlada su posición según la posición de la mano correspondiente.
- listaDibujos, lista que contiene los objetos de la clase Dibujo que han sido dibujoEnMovimiento y el usuario ha dejado de mover al cruzar los brazos.
- objetoAleatorio, es un objeto de tipo Random que sirve para generar números aleatorios.
- familiaDibujos, guarda el String con el nombre de la familia de imágenes con la que está jugando el usuario en ese momento.

Esta clase será la que trate los eventos lanzados por la clase KinectManager, para cada evento se ha creado un método que dará una respuesta a este evento.

- _kinectManager_cogerDibujo, se ejecuta cuando se lanza el evento CogerDibujo, este método comprobará que en la variable ladoConDibujo el ladoConDibujo sea none (es decir, que no haya ningún lado estirado), es ese caso sobrescribirá esa variable con por el del brazo que ha sido estirado y según sea el lado estirado creará un objeto Dibujo pasándole como posición las coordenadas de su mano izquierda o derecha. También se llamará al método LoadContent del objeto Dibujo.
- _kinectManager_dejarDibujo, al lanzar el evento DejarDibujo este método se ejecuta, lo primero que hace es comprobar que la variable ladoConDibujo sea distinto de none (es decir, que haya un brazo estirado) en caso afirmativo se guardará el dibujoEnMovimiento en la listaDibujos y posteriormente se igualará a null. Por último la variable ladoConDibujo se le dará el valor none.
- _kinectManager_nuevoJugador, cuando KinectManager detecta un nuevo jugador lanza el evento y se ejecuta este método. Durante la ejecución de este método la variable dibujoEnMovimiento se iguala a null y listaDibujos es instanciada de nuevo (vuelve a estar vacía) de esta manera en la nueva pantalla no aparecerá ningún dibujo. Por último la variable ladoConDibujo se le dará el valor none. Además se desea que el nuevo usuario trabaje con una nueva familia de imágenes, para ello con la ayuda del objeto objetoAleatorio se generarán números aleatorios hasta que se genere un número que este asociado a una familia de imágenes (en el diccionario) distinta a la familia con la que se estaba trabajando en ese momento.
- calculoCambioRotacion, este método calculará el ángulo de inclinación de la recta que une las dos manos. Este ángulo será utilizado para la rotación del objeto dibujoEnMovimiento. Para calcular el ángulo se calculará la pendiente de la recta (que se corresponde con la tangente del ángulo) y con el método Math.atan se obtendrá el ángulo en radianes.

El resto de métodos se corresponden con los métodos propios de XNA (aunque sólo se van a describir aquellos sobre los que se ha realizado alguna modificación.

- Game1, es el constructor, aquí se establecerá el tamaño de la pantalla y se instanciará el objeto _kinectManager, también se ligarán los eventos lanzados por este con los métodos descritos anteriormente. El campo listaDibujos será instanciado aquí.
- LoadContent, se cargará la imagen de fondo y se llamará al método LoadContent de _kinectManager.
- Update, en el caso de existir un dibujoEnMovimiento (que sea distinto de null) se llamará al método Update de éste pasándole las coordenadas de la mano derecha o izquierda (según el valor del campo ladoConDibujo). También se calculará el ángulo de inclinación de la recta que forman las dos manos (llamando al método calculoCambioRotacion) y será pasado al objeto dibujoEnMovimiento a través de su propiedad modificacionRotacion.
- Draw, dibujará los diferentes elementos en el siguiente orden:
    - Fondo de pantalla, según el color guardado en colorPantalla.
    - Se llamará al método Draw de _kinectManager para que dibuje la sombra del jugador en caso de ser necesario.
    - Para dibujar cada uno de los objetos Dibujo que ya no están en movimiento se llamará al método Draw de cada elemento de la lista listaDibujos.
    - Si dibujoEnMovimiento es distinto de null se llamará a su método Draw para que sea dibujado.

- UnloadContent, se llamará al método UnloadContent de _kinectManager que desiniciará el dispositivo Kinect y pondrá el grado de orientación de su cámara a cero.

Para que el campo colorPantalla pudiera ser accesible desde el exterior (en concreto desde _kinectManager donde se sigue la secuencia de colores a pintar) se ha creado la propiedad ColorPantalla que permite desde el exterior acceder y modificar esta variable.

Capítulo 7 Conclusiones y líneas futuras

A lo largo del desarrollo del proyecto se ha realizado una primera aproximación a la herramienta Kinect y se ha comprobado que sus campos de aplicación van mucho más allá que el campo de los videojuegos para el que estaba originalmente diseñado.

La primera parte del proyecto consistió en una aproximación a la herramienta y su SDK, durante esta fase se realizaron cuatro mini-aplicaciones con el objetivo de familiarizarse con la plataforma y tener una primera idea de las posibilidades de la herramienta. Con este conocimiento ya era posible pensar en posibles campos donde el uso de Kinect pudiera ser útil e innovador.

Uno de estos campos era el campo de la educación. Se deseaba desarrollar un programa que ayudara en el aprendizaje de tareas cotidianas, el objetivo era aprovechar las nuevas opciones de interacción que ofrece Kinect con el fin de conseguir una mayor motivación e implicación por parte de los usuarios. Finalmente se desarrolló una aplicación que simulaba la tarea de poner la mesa.

Tras las pruebas con los usuarios se pudo comprobar que en efecto, aunque Kinect no permitía una interacción perfecta, había una gran motivación por realizar la actividad por parte de los usuarios. Tras estas pruebas se llegó a la conclusión de que si que era posible el uso de esta herramienta para propósitos educativos, los usuarios mostraron un gran interés por la herramienta y la forma de interacción que se planteaba.

Esta primera experiencia ha sido satisfactoria y da pie a futuros desarrollos en colaboración con asociaciones especializadas con el objetivo de enfocar esas futuras aplicaciones a sectores más concretos. Así mismo esta misma aplicación se podría desarrollar añadiéndole más funcionalidades, incluso se podría unir a otros juegos que simulen actividades cotidianas (preparar la mochila del colegio, la ropa, etc) creando un cuaderno de ejercicios, además aprovechando la arquitectura modular que se ha dado a la aplicación (las clases que entran en contacto con Kinect están aisladas) sería posible probar su funcionamiento con otros SDKs o dispositivos similares a Kinect (como el de ASUS).

El sistema, su descripción, las pruebas y sus resultados y conclusiones forman parte de un paper que ha sido presentado al congreso internacional **Interaction Design and Children 2012** que se celebrará en Bremen, Alemania, el mes de junio.

Paralelamente Isabel Sánchez Gil, planteó la posibilidad de incluir en su futura exposición “Cuando el umbral diferencial no es visible” (ver apéndice dos), atraída por la nueva forma de interacción que ofrecía la herramienta, una aplicación que utilizara Kinect. Los visitantes a la exposición podrán interactuar con trozos de su obra con el objetivo y hacer su propia composición a partir de estas. El objetivo es que éstos puedan mostrar su propia interpretación de la realidad.

Personalmente, veo en Kinect una herramienta con infinidad de campos de aplicación, es una tecnología reciente y debe mejorar mucho algunos aspectos técnicos, cada vez hay más equipos trabajando con esta tecnología y las mejoras del SDK oficial son periódicas, en el plazo de un año es muy probable que veamos un avance muy importante en la cantidad y calidad de los desarrollos.

Además la realización del proyecto me ha dado, aparte de la experiencia adquirida en el campo del desarrollo con Kinect, la posibilidad de trabajar en un proyecto innovador en el que he tenido que probar diferentes técnicas y soluciones barajando los resultados obtenidos para conseguir la mejor solución al problema planteado. Como resultado de este proyecto he tenido la posibilidad de colaborar en la creación de una publicación para el congreso internacional anual IDC.

Por otro lado, la colaboración con Isabel Sánchez me ha permitido utilizar mis conocimientos adquiridos en un ámbito nuevo para mí como es el mundo del arte. Además de darme la posibilidad de crear un aplicación siguiendo los requisitos marcados por alguien que no tiene ningún tipo de contacto con la informática.

Capítulo 8 Bibliografia

El LENGUAJE DE PROGRAMACION C# (2001), José Antonio González Seco

AUDIO: PROGRAMACIÓN EN AMBIENTES WINDOWS, Luis Diego González Zúñiga

WINDOWS PRESENTATION FOUNDATION (2009(, Pedro Bauzá Picó

PROGRAMING GUIDE: Getting Started with the Kinect from Windows SKD Beta from Microsoft Research (2011).

http://msdn.microsoft.com/es-es/library/kx37x362%28v=VS.80%29.aspx

http://channel9.msdn.com/

http://elrincon-delgamer.blogspot.com/2010/08/caracteristicas-y-funcionamiento-de.html

http://blogs.msdn.com/b/eternalcoding/archive/2011/07/04/gestures-amp-kinect.aspx

http://msdn.microsoft.com/es-es/aa937791

http://lex0712.xnalatam.net/2011/03/02/serie-de-tutoriales-xna/

http://create.msdn.com/en-US/education/gamedevelopment

Anexo 1 Home videogame devices like Kinect as educational tools for children with special needs

I.Iralde
Public Univ. of Navarra
Campus Arrosadia, s/n
31006 Pamplona, Spain
+34 948 169530
iralde.53259@e.unavarra.es

I. Sanchez
ANFAS
C/ Pintor Maeztu, 2
31008 Pamplona, Spain
+34 948 153044
isabelsanchezgil@yahoo.es

B. Bossavit
Public Univ. of Navarra
Campus Arrosadia, s/n
31006 Pamplona, Spain
+34 948 169538
benoit.bossavit@gmail.com

A. Pina
Public Univ. of Navarra
Campus Arrosadia, s/n
31006 Pamplona, Spain
+34 948 169538
pina@unavarra.es

ABSTRACT

During the last months we have started collaboration with parents belonging to an association which supports families with Autistic Spectrum Disorder (ASD) children in the area of Navarra, Spain. In this paper we describe the experience of designing, implementing and evaluating a Kinect-based digital tool for learning how to lay the table. This tool has been tested with 3 children, one of them with ASD. The main goals of our work were on the one hand to show that we can use home videogame devices for educational purposes, and on the other hand to create a working group for carrying out such developments. We have established an interdisciplinary team in where professionals, parents and computer scientists are collaborating for providing appropriate digital tools for ASD children. These tools are intended to be used at home with the family.

Categories and Subject Descriptors

K.4.2 [**Computers and Society**]: Social Issues – *Assistive technologies for persons with disabilities.*
H.5.2 [**Information Interfaces and Presentation**] User Interfaces – *User-centered design.*

General Terms

Design, Experimentation, Human Factors.

Keywords

Social skills, daily activities, autism spectrum disorders, general communication disorders, analogical vs digital tools.

1.INTRODUCTION

Many families with Autism Spectrum Disorders (ASD) children have to receive specific training in order to be able to give and answer to their children needs.

IDC '12, June 12-15, 2012 Bremen, Germany

The Autism section of ANFAS (association of Navarra in favor of people with intellectual impairment, http://www.anfasnavarra.org/) has been training not only parents but also professionals in charge of stimulating their children with ASD.

From the Math and Computer Engineering department of our university we have started collaboration with some families belonging to the Autism section of ANFAS, which have been trained by DELETREA (Psychology and language center of Madrid, http://www.deletrea.es/). The purpose is to bring new technologies and the stimulation processes used with ASD children into close proximity to potential recipients. In particular we have focused on the digitalization of analogical tools that both, professional and families, use with the aim of optimizing the results during the interventions.

In this paper, we are trying to assist the already trained parents who want to also stimulate their children at home.

Among all the existing references that most professionals and parents are using, we would like to highlight some of the "Monfort Handbooks" [9] to work with the verbal language (augmentative) by means of some activities such as scene description for improving conversational or stimulating verbal memory skills. The proposed methodology is based on a provided set of materials but also it encourages parents to develop their own materials (analogical ones) to meet the particular needs of every individual and family.

On the one side we are currently working on communication aspects (comprehension and expression) and on the other side daily basic skills and competences. In both cases families work with different, most of the time, analogical tools. For the purpose of digital tools design, we are using natural interaction techniques and appropriate technologies to practice these activities at home with the family.

Our main goal is to develop and improve these digital tools with the above mentioned ideas and the optimal environment. The target of this paper is to validate the use of the Kinect videogame device in order to develop a digital tool which makes use of natural interaction and is suitable to be used at home.

Along with the usual characteristics, we have to deal with a special kind of user profile (see background below). We are also working implicitly on improving aspects like social interaction or communication, active participation promotion, users' motivation, using the system for in situ trials and giving immediate feedback whenever it is possible.

Our contribution to this paper is to show how we can get hold of using close and familiar devices currently being used in games and to use them for our educational purposes.

Some of the questions that arise to us are:

- Is the system a workable solution to learn "how to lay the table"?
- What kind of interaction should we use: hands, voice or both?
- Are the children able to use the body/hands/voice to interact?
- Are the children motivated to use such system?
- What are the advantages of using digital tools for these processes (instead of analogical tools as shown in figure 1)?
- Why this kind of tools and working methodologies are especially appropriate for children with special needs?

Figure 1. The paper-based tool to learn "how to lay the table"

The rest of the paper is organised as follows: the next section outlines the most representative work related to our ideas. Then we propose a prototype that makes our objectives attainable, including a preliminary evaluation and discussion. We finish with some conclusions and future lines.

2.BACKGROUND

During the last 20 years, a lot of researchers have demonstrated that computer applications can help children with special needs. The European project, COSPATIAL (http://cospatial.fbk.eu/), presents an exhaustive State of Art about computer science and autism from papers gathered until the year 2009. In their work, it is possible to find a list of scientific research papers and their description organized in five categories: social skills training, video modeling, virtual reality, shared surfaces and robotic for ASD. In this section we present only the latest works related to our objectives.

As we stated previously in the introduction, training socials skills is one of the main axes in research for children with special needs. Many works in the literature try to improve specific skills among social skills.

On the one hand, some researchers try to improve the communication skills by understanding and

detecting people emotions. In [6] they present a multitouch application to deform a face that allows the child to create by himself an emotion. Inversely, using video processing, in [8] they propose a technique which extracts emotion from a video recorded face.

On the other hand, the collaboration skill is an active axis of investigation too. In [2] they propose a collaborative puzzle game which encourages collaborative gestures. They compare the behavior between children and children with ASD. The result shows that children with ASD need more moves and coordination. In [12] they propose series of collaborative applications. These mini games force the children to collaborate and synchronize with their partner if they want to progress in the game. In this study they analyze through the therapists adaptation, efficacy, playability and motivation of children with ASD.

There are many other social skills to improve. We can present them as daily activities. In [4] they present a visual schedule installed in a classroom. A visual schedule can remove anxiety, help in the organization of the time and language comprehension. In this case, each child works with a pocketPC for seeing all the activities and completing them. In [11] they present two virtual reality applications to help the child inside a supermarket and a school. In each case, the child can explore the environment and perform active tasks (e.g filling the shopping list) or stimulate his imagination by seeing a short video animation.

Focusing on the technologies that are used, most of the works use multitouch tables, pocketPCs or tablets as devices. In [3] and [5] they compare mouse-based and tactile-based interfaces and show that mouse is more precise but tactile is easier and more efficient. In [1] and [10] they compare mouse-based task with tangible interface and show that children perform the same task faster with tangible interface.

We have not found any reference in the literature that works with daily activities to be used at home with the family. Furthermore we can realize that few research studies try to relate the Kinect device with education and even less with children with ASD. In [7] they show the possibilities of the Kinect for classroom interaction. We can find also on the internet some examples of children with special needs using the Kinect device [13][14][15].

3.One example: using Kinect to learn “how to lay the table”

System description

The Microsoft Kinect is a low-cost tool which allows the user to interact using his own body. Our prototype consists in using this device as interaction input to learn “how to lay the table”. As we said before, parents or professionals generally use an analogical method to perform daily tasks. Therefore, with this experiment we want to know if a digital counterpart can motivate and help children with ASD.

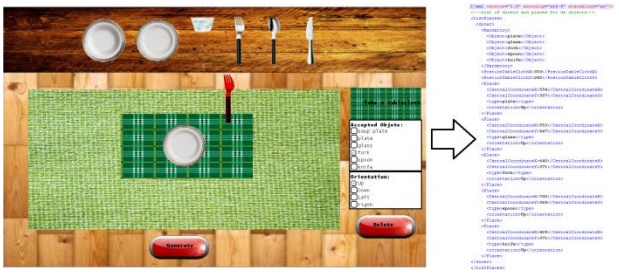
The study is performed with a regular computer under Windows 7, connected to a projector for the visualization and to the Microsoft Kinect tool.

Figure 2. Mr Brown is grasping a glass

The aim is to take objects (plates, knives, forks, etc…) and place them on the table (cf. Figure 2). The system brings some additional information like the position of the hands or the shadow of the element when the user is close to the right position. Actually the system knows the correct position of each item as it was previously set by the parents or professionals. This configuration is created through a graphical interface (cf. Figure 3) and stored in an XML file. The game ends once the system checks that the table is complete.

The interaction must be easy, natural and have a short adaptation time. Thus our first idea was opening and closing the hand to take and release objects; nonetheless it is a gesture that the Kinect SDK cannot easily detect. We tested two ways of interaction: one using both hands and the other one mixing hand and voice. In the first case the child must put his both hands around the item to select it and has to separate them to release it. In the second case the child must talk to select ("take") and release ("leave") the object, one hand is used to move the element, the hand cursor has to be located over the object when the user talks in order to take/leave it.

Figure 3. Graphical interface to configure a session



Experience description

We tried the application with three children. For privacy reasons we give fake names to the users: a 6 years old girl, Mrs Pink; a 10 years old girl, Mrs Orange; and an 11 years old boy with ASD, Mr Brown.

The experiment was performed separately: every child entered the room with their parents and remained inside when the intervention finished. As a result each child used the application without previous knowledge or clues observed from other users. All the sessions were recorded with two cameras, one in front of the child and other facing the display. Firstly, we let the child try the mini-game without explanations in order to observe his reaction and motivation using the system, and then we gave him more information about how to correctly interact.

We always started with the two hands interaction and next proceeded with the voice and hand interaction.

The overall experiment was done in about 45 minutes and every child used the system for a time varying from 5 to 10 minutes.

Discussion

In this part, we organize our observations and interpretations about the system around two axes: interaction and motivation.

We are pleasantly surprised with the two hands interaction as the three children quickly found the right gesture without our help. Mrs Pink and Mrs Orange reached the objective with no particular problems. Nevertheless Mr Brown had more difficulties manipulating the object. Actually one limitation of the skeleton recognition algorithm is the instability occurring when the user put his hands too close. That caused a real problem to Mr Brown as he wanted to join his hands even when we recurrently warned him. The main consequence was an inopportune drop of the selected object or an exchange if another element was placed under the current position. This can be a real inconvenience; one solution would be a limitation of freedom and only allow the user to release the object on its initial or correct place.

Regarding the second way of interaction the SDK provides speech recognition only in English, for that reason we needed to explain how to pronounce the keywords and help them during the session.

Abstracting the linguistic problem, we could notice a real improvement of Mr Brown, the completion time was three times less than with the two hands method. Contrary to what one might think, Mrs Pink and Mrs Orange needed twice more time to reach the objective. This may be due to the fact that for being heard or understood by the system the user had to speak loudly, they were unable or ashamed.

These are usual problems of voice recognition software: language compatibility, need of quiet place or processing time. One solution would be to find a more efficient library compatible with Kinect.

To summarize, Mr Brown prefers the voice interface contrary to Mrs Pink and Mrs Orange who prefer the two hands interface. A mix of both interfaces seems difficult to implement, therefore a good compromise would be a menu to select the interaction technique before or during the play session.

From this experience we can analyze two important aspects: motivation and perseverance.

Mr Brown took the initiative coming near the Kinect to observe the system and he said to us “this is the Kinect!”, he does not have one at home but he knows what it is owing to television advertisement.

We can notice real perseverance from the three children. During their turn, they untiringly tried to reach their objective even if the system caused inopportune object exchanges.

Mr Browns’s mother was present during the experiment and gave us some feedback. First of all and for her, the main interest is not to learn how to lay the table but to continue working on the language issues, verbal and semantic understanding and expression. We believe that every possible user will have specific needs and objectives when using the system.

Another important issue is about motivation and for that she was clear: game devices and similar gadgets get Mr Brown motivated. We also observed that.

At this point she was explaining the way she has been using the analogical tool: normally at home (not at school) and just before starting the meal. Mr Brown was motivated as it was not only a game and the result of the simulation had to be applied in order to lay the “real” table for the “real” meal. Our system is compatible with such working method.

The user’s experiences have been individual ones; nevertheless at some point and in a spontaneous way Mr Brown started helping Mrs Pink when she was trying to interact using the voice commands. He tried to help her in pronouncing the right world in English. This fact made us think that it would be possible to add a collaborative component to the system in order to involve in the gameplay parents, therapists, friends or brothers and sisters.

4.Conclusion and future lines

This paper shows a preliminary work on the fact of using Kinect to design and implement digital tools for children in order to manage daily activities learning.

Based on the observations during the evaluation we conclude that it is possible to use such a game device for a serious education game. We can answer to some of the questions stated at the beginning: we should maintain both ways of interaction, children are able to interact without any or very little help and they are very motivated for using such systems.

Other questions have only been partially addressed and represent future lines to work on.

The tool is suitable for children, nonetheless for ASD children it is important to add some specific features to promote verbal communication, comprehension and expression during the activity.

Another issue to be addressed in a deeper way is the evaluation methods for measuring the performance of the applications as education tools. These methods will help in choosing which tool, analogical or digital, is more suitable. Digital tools have the usual advantages: it is easy to share them, make copies, extend or integrate multimedia content. Nevertheless the processes by themselves seem similar from the point of view of the therapists.

6.ACKNOWLEDGMENTS

We would like to acknowledge ANFAS and the families that have participated with us in the design, implementation and evaluation of our prototype.

This work has been partially supported by the EU Alfa project "Gaviota" (DCI-ALA/19.09.01/10/21526/245-654/ALFA 111(2010)149)

We would like also to acknowledge our colleagues Asier Marzo and Oscar Ardaiz for their suggestions and help with English.

7.REFERENCES

[1]. Antle, A.N Droumeva, M. And Ha, D. 2009. Hands on What? Comparing Children's Mouse-based and Tangible-based Interaction. *Proceedings of IDC 2009, ACM Press (2009), 80-88*

[2]. Battochi, A. Ben-Sasson, A. Esposito, G. Gal, E. Pianesi, F. Tomasini, D. Venuti, P. Weiss, P.L. and Zancanaro, M. 2010. Collaborative Puzzle Game: a Tabletop Interface for Fostering Collaborative Skills in Children with Autism Spectrum Disorders. *Journal of Assistive Technologies*. 2010, 4(1):4:14

[3]. Forlines, C. Wigdor, D. Shen, C. and Balakrishnan, R. 2007. Direct-Touch vs. Mice Input for Tabletop Displays. *Proceedings of CHI\"07, ACM Press (2007), 647-656*

[4]. Hirano, SH. Yeganyan, M.T. Marcu, G. Nguyen, D.H Boyd, L. And Hayes, G.R. 2010. vSked: evaluation of a system to support classroom activities for children with autism. *In: Proceedings of CHI 2010, pp 1633–1642*

[5]. Hornecker, E. Marshall, P. Sheep Dalton, N. and Rogers, Y. 2008. Collaboration and Interference: Awareness with Mice or Touch Input. *Proceedings of CSCW\"08, ACM Press (2008), 167-176*

[6]. Hourcade, J.P. Bullock-Rest, N.E. and Hansen, T.E. 2010. Multitouch Tablet application and activities to enhance the social skills of children with autism spectrum disorders. *Journal of Personal and Ubiquitous Computing. 2010.*

[7]. Hsu, J.H. 2011. The Potential of Kinect as Interactive Educational Technology. *2nd International Conference on Education and Management Technology, IPEDR vol.13 (2011) © (2011) IACSIT Press, Singapore*

[8]. Madsen, M. El Kaliouby, R. Goodwin, M. And Picard, R. 2008. Technology for just-in-time in situ learning of facial affect for persons diagnosed with an autism spectrum disorder. *In: Proceedings of ASSETS 2008, pp 19–26*

[9]. Monfort, M. and Monfort, I. 2001. En la mente 2: un soporte para el entrenamiento de habilidades pragmáticas en niños: como decirlo. *Entha ediciones, 2001.*

[10]. Sitdhisanguan, K. Chotikakamthorn, N. Dechaboon, A. and Out, P. 2011. Using tangible user interfaces in computer-based training systems for low-functioning autistic children. *Journal of Personal and Ubiquitous Computing, 2011.*

[11]. Vera, L. Campos, R. Herrera, G. and Romero, C. 2007. Computer graphics applications in the education process of people with learning difficulties. *Journal of Computers & Graphics 31 (2007) 649 – 658*

[12]. Weiss, P.L. Gal, E. Eden, S. Zancanaro, M. and Telch, F. 2011. Dimensions of Collaboration on a tabletop interface for children with Autism Spectrum Disorder. *Proceedings of CHI 2011.*

[13]. http://www.kinecteducation.com/, Kinect and Education, March 2012.

[14]. http://www.ntu.ac.uk/apps/news/ui/PrinterFriendlyStory.aspx?sysLocat=15&sid=109137, Game devices for ASD, March 2012

[15]. http://lakesideautism.com/, Autism center, March 2012

Anexo 2 Isabel Sánchez, su obra y Kinect

"Cuando el umbral diferencial no es visible".

Del 27 de abril al 10 de junio del 2012 se expone en la Sala El Polvorín de la Ciudadela de Pamplona. Organizada por el ayuntamiento de Pamplona.

"Otra mirada"

El uso de "kinect", una herramienta que posibilita el desarrollo de aplicaciones que permiten controlar e interactuar sin necesidad de tener contacto físico, reconociendo posturas y objetos así como captando imágenes ha permitido a Isabel Sánchez Gil reflexionar y mostrar que hay otras formas de interactuar con la realidad, percibiéndola de diferente manera y por lo tanto llegando al conocimiento desde otras formas de interacción. En su última obra (**"*Otra mirada*"** presentada dentro de la exposición **"*Cuando el umbral diferencial no es visible*"** en la Sala el Polvorín de la Ciudadela de Navarra desde el 27 de Abril hasta el 10 de junio del 2012) la aplicación desarrollada permite sugerir que la realidad es diferente para cada uno de nosotros y nos invita a la participación, donde el espectador se convierte en actor y forma su propia interpretación a partir de obras expuestas y fragmentos de otras. Este sistema le ha permitido continuar en su indagación de nuevas formas de expresión plástica.

Isabel Sánchez Gil, artista multidisciplinar, afincada en Navarra, Diplomada en Educación General Básica y licenciada en Bellas Artes en la especialidad de pintura por la Universidad de Barcelona. Paralelamente se formó en la escuela de Artes y Oficios, Diputación de Barcelona, en la especialidad de grabado. Participó en talleres organizados por Quam-Associacióper a les ArtsContemporánies en Lleida: Uno dirigido por Joan Rom en 1994 y otro dirigido por Antonio Muntadas(Noviembre del 95/Mayo del 96).Posteriormente se ha formado en creación y edición audiovisual. Sigue formándose dentro del campo de las nuevas tecnologías. Está vinculada a la docencia en Educación Primaria y Secundaría. Ha tenido la oportunidad de investigar y reflexionar sobre los trastornos del desarrollo del lenguaje, creando otras formas de comunicación, conjugándolas con la comunicación verbal, abriendo otros caminos para poder comprender y relacionarse con el mundo.

Participa como asesora externa en varios proyectos de la Universidad Pública de Navarra relacionados con nuevas tecnologías, trastornos en la comunicación y formas de interacción; esto le llevó a participar en la presentación de un comunicado en el Congreso Internacional de EDUTEC 2010 Titulado "Nuevas tecnologías y trastornos del desarrollo".

Durante el curso 2010-2011 concluyó el Master oficial *Artes Visuales y educación: un enfoque construccionista* convocado por la facultad de Bellas Artes de la Universidad de Barcelona, dirigido por Fernando Hernández.

Ha participado como ponente en el I Congreso Síndrome up, Pamplona 2011 con la comunicación *"Los cambios de mi mirada"*

Actualmente compagina su función como Profesora de Educación Visual y Plástica en Educación Secundaria con su labor investigadora dentro de los cursos de doctorado *Arte y Educación* en la Universidad de Barcelona. En su tesis doctoral se plantea el papel de las Artes en la Educación. Las Artes como punto de partida para entender mejor el mundo, dando la posibilidad al alumnado de desarrollarse como persona, proporcionándole estrategias para desenvolverse en el mundo donde viven y sobre todo indagar herramientas para adquirir una actitud investigadora donde el alumnado vaya creando sus conocimientos.

Dirige y coordina un grupo de trabajo formado por maestros de primaria con el objetivo de profundizar en el campo de las Artes Visuales en la escuela.

Como hilo conductor de su investigación plástica y visual se cuestiona cómo se adquiere el conocimiento, cuestionándose también la realidad y llegando a conclusiones parciales (siempre revisables) como: ***la realidad es una realidad parcial***, una realidad percibida por cada uno de nosotros.

Su objeto de investigación ha estado siempre relacionado con las diferentes formas de comprender el mundo que nos rodea, diferentes formas de conocer, diferentes formas de interpretar. Se pregunta dónde se encuentra ese punto de inflexión, donde las etiquetas y el lenguaje como elemento que permite el conocimiento deja fuera otros *lenguajes*, deja fuera todo aquello que no se nombra. Estas múltiples pero enlazadas inquietudes le ha llevado por cambiantes recorridos que ha revisado desde diferentes puntos de vista.

En su incesante búsqueda de un código apropiado para expresar plásticamente sus cuestionamientos ha pasado por el dibujo, la pintura, la fotografía, el video y otras nuevas tecnologías que le están permitiendo y abriendo nuevas posibilidades.

Desde sus dudas de cómo creamos conocimiento, cómo llegamos a la comunicación, cómo utilizamos el lenguaje ha desembocado en el mundo de las nuevas tecnologías. Las nuevas tecnologías como herramientas que permiten un aprendizaje más eficaz, pero sin dejar de cuestionarse esta eficacia tecnológica dentro de nuestra sociedad.

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

Desarrollo de aplicaciones con Microsoft Kinect

Autor: Iñaki Iralde

Tutor: Alfredo Pina

Desarrollo de aplicaciones con Microsoft Kinect

- Aproximación a la herramienta Microsoft Kinect
- Ver posibles campos de aplicación
- Desarrollo de aplicaciones en alguno de estos campos

Desarrollo de aplicaciones con Microsoft Kinect

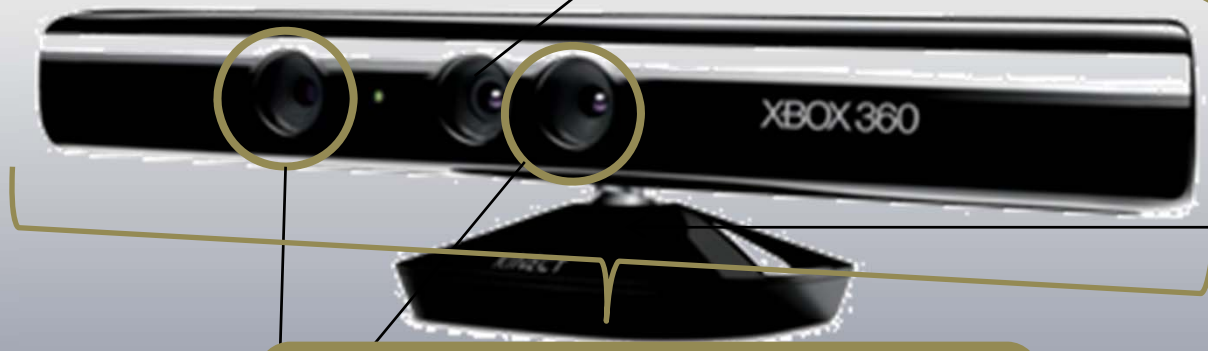
- Dispositivo de control.
- Interacción natural con el cuerpo
- Aparición noviembre 2010
- Junio 2011, aparición SDK beta
- Febrero 2012
  - Kinect para Windows
  - SDK comercial

Desarrollo de aplicaciones con Microsoft Kinect

- Memoria RAM 512mb
- Acelerómetro
- Ventilador

Cámara RGB

- 640x480
- 30 fps



Motor de inclinación

Micrófono multi-array:
4 micros → 1 sólo

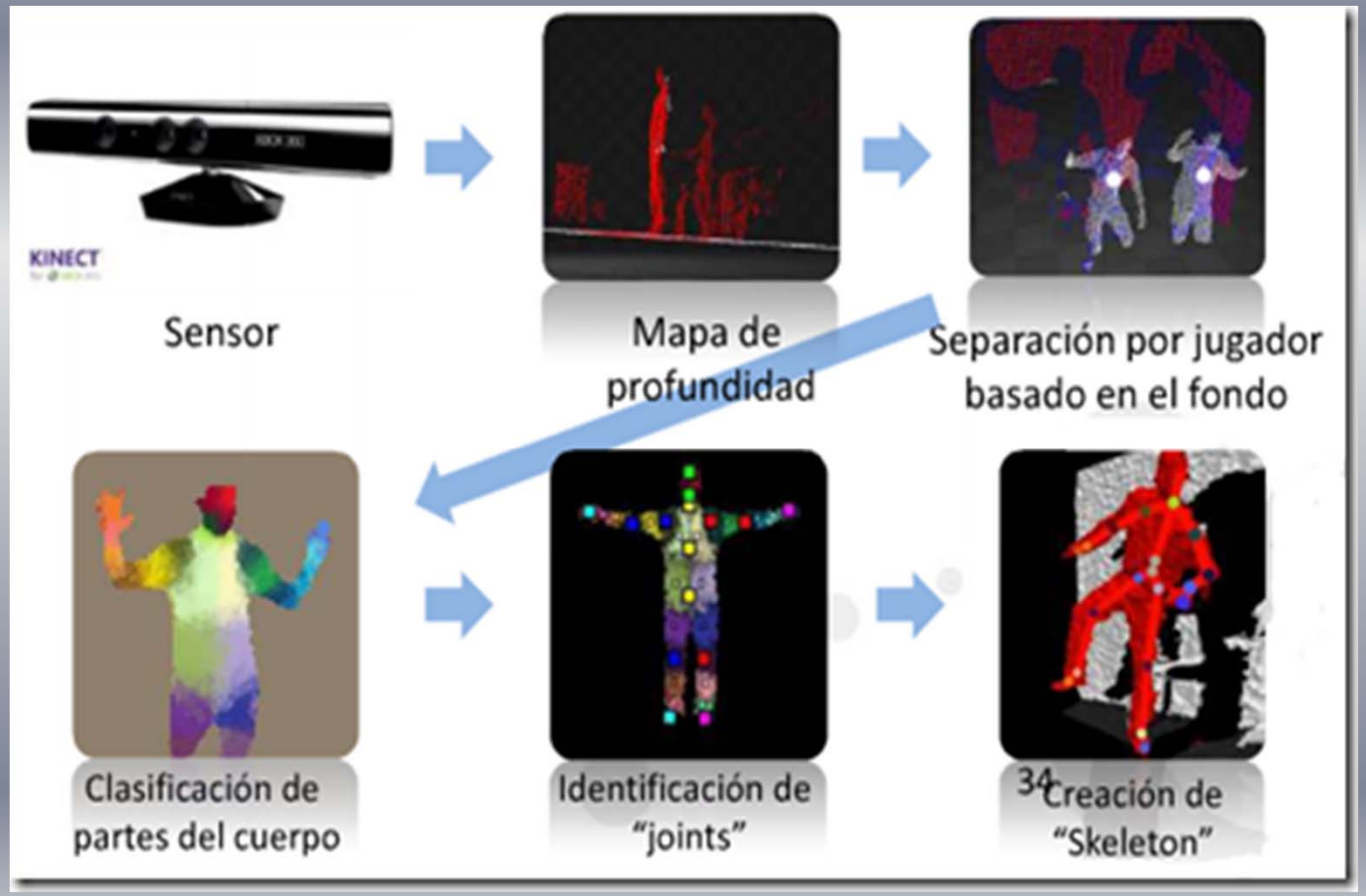
Se… profundidad

- Retícula derecha: … Proyector … sensor

Desarrollo de aplicaciones con Microsoft Kinect

- Tecnología no compleja y antigua
  - *Disponible a bajo coste*
- Rebote haces laser
  - *cálculo profundidad*
- Filtros
  - *Saber qué es un jugador y qué no*
- Kinect dispone de 200 posiciones cargadas
  - *Si se tapa parte del jugador, recurre a ellas.*

Desarrollo de aplicaciones con Microsoft Kinect



- Evitar otras voces y ruidos ambientales
- Tras estudio en diferentes hogares
  - *Diseño actual de los micrófonos*
- Detección del origen por software
- Ejecutándose continuamente

- Diferentes disposiciones de los espacios de uso

Dar movimiento a la cámara

+28º

-28º

Desarrollo de aplicaciones con Microsoft Kinect

- SDK Beta2, licencia no-comercial
- Orientado a la investigación académica
- Desarrollo en C# y C++
- Desarrollo bajo Visual Studio 2010
- Solamente utilizable con Windows 7
- Información captada accesible
  - *Polling* Mode
  - *Event* Mode

Desarrollo de aplicaciones con Microsoft Kinect

Rango de Profundidad	
Sensor de color y profundidad	De 1,2 metros a 3,5 metros
Detección de esqueletos	De 1,2 metros a 3,5 metros

Especificaciones del sensor Kinect	
Angulo de visión	43º de campo de visión vertical 57º de campo de visión horizontal
Angulo de movimiento del motor	+28º y -28º
Velocidad de frames	30 frames por segundo (FPS)
Resolución (profundidad)	QVGA (320 x 240) 16bits
Resolución (Cámara RGB)	VGA (640 x 480) 32 bits
Formato de Audio	16-kHz, 16-bit PCM
Características de entrada de Audio	Cuatro micrófonos en array con un convertidor analógico a digital de 24 bits (ADC) y el procesamiento de señales de Kinect con cancelación del eco y eliminación de ruido de fondo.

Desarrollo de aplicaciones con Microsoft Kinect

- Cada pixel de la imagen ➡ 4 bytes

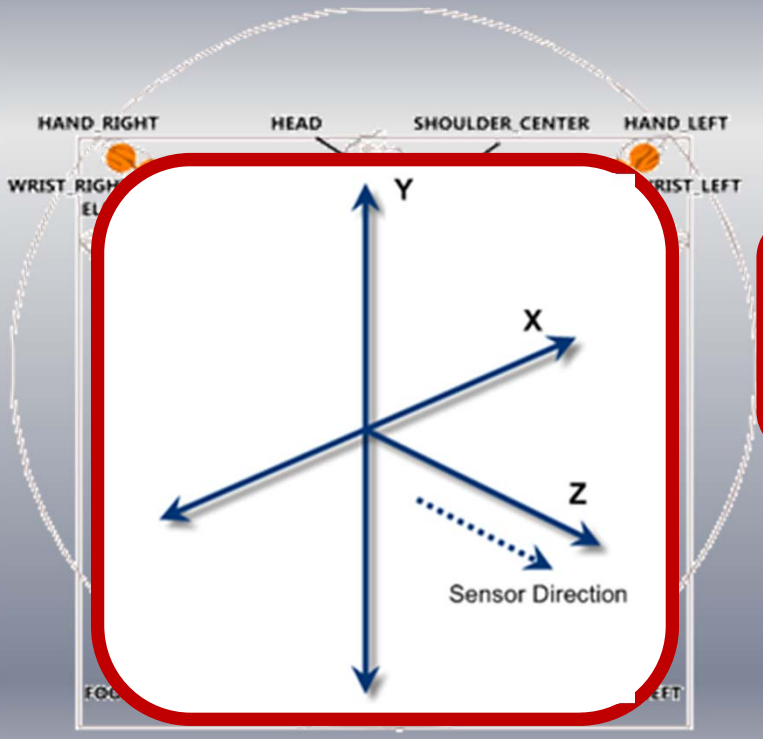
- Imágenes de tamaño 640x480
- Cuando se quiere tratar con imagen RGB:
  - Abrir flujo de color (tamaño, tipo imagen…)
  - Asociar el evento que se lanza cada nuevo *frame* (donde se tratará la imagen).

Desarrollo de aplicaciones con Microsoft Kinect

- Frames de 320x240 pixeles. Cada pixel ➡ dos bytes

Desarrollo de aplicaciones con Microsoft Kinect

- Localización de puntos del cuerpo o Joints

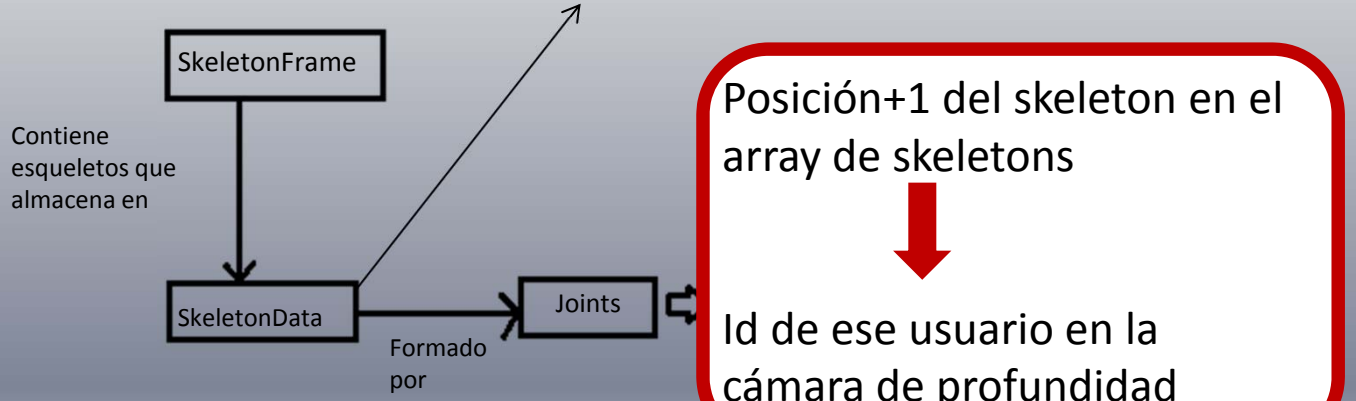


Coordenadas indican distancia en metros desde el Kinect

Desarrollo de aplicaciones con Microsoft Kinect

- tipos tracking
  - activo, posición y Joints,2 skeleton
  - pasivo eleton

Cada skeleton tiene su propio TrackingId



Detección de gestos:

- Seguir la evolución de la posición de Joints en el tiempo
  - *Usando una lista como estructura de datos*
- Cada nuevo frame, comprobar si:
  - Posición ha cambiado lo suficiente

    en intervalo de tiempo dado.

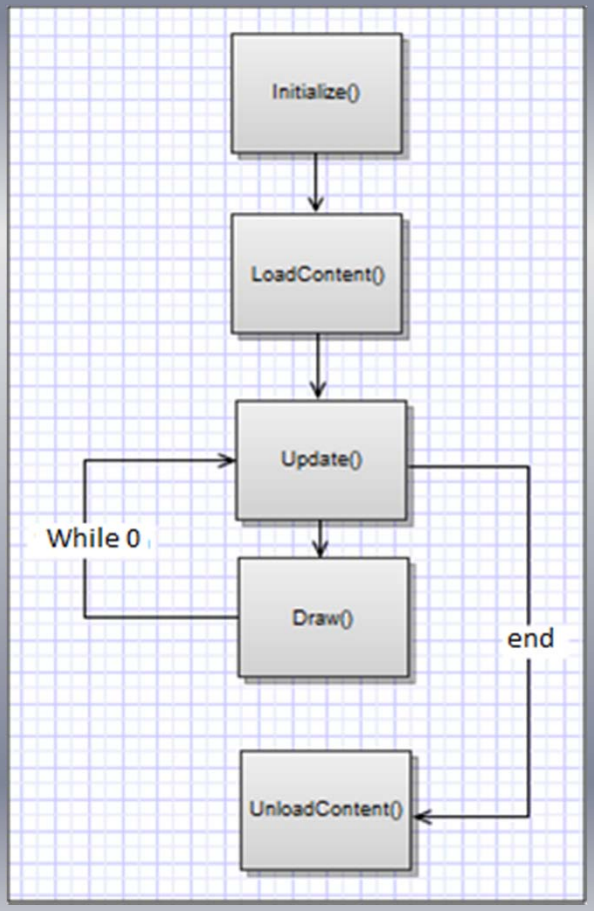
Desarrollo de aplicaciones con Microsoft Kinect

- Funcionando continuamente
- Sólo en Ingles
- Reconoce patrones de palabras

- Inconvenientes
  - Todavía muy sensible a ruidos externos
  - Tiempo de procesamiento

Desarrollo de aplicaciones con Microsoft Kinect

- Framework creado para el desarrollo de juegos
- Sencillez uso de imágenes, colisiones…
- Bucle del juego ya implementado
- Facilidad para en un futuro migrar a plataforma Xbox360



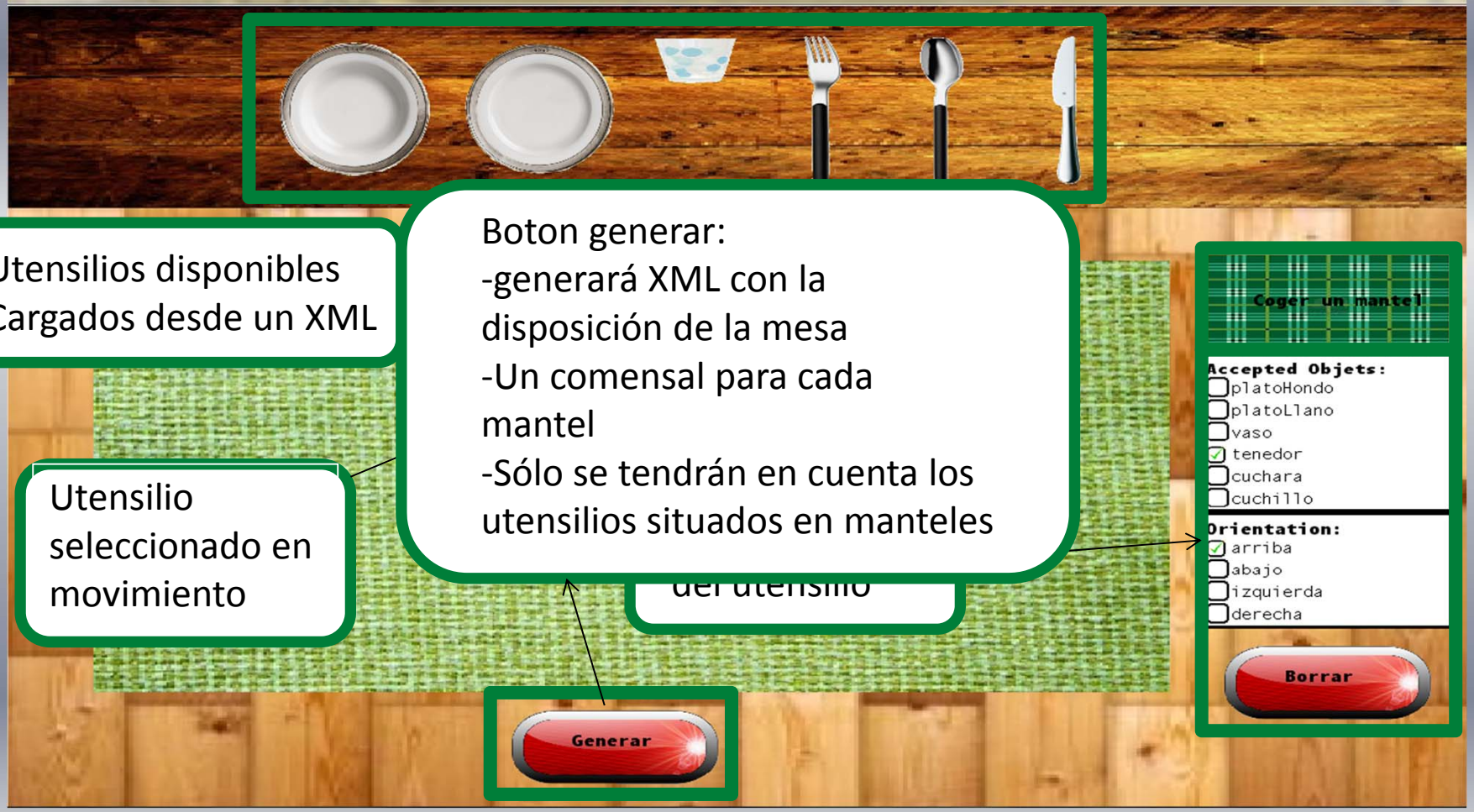
Desarrollo de aplicaciones con Microsoft Kinect

- Prototipo de herramienta:
  - Bajo coste
  - Para el aprendizaje de tareas cotidianas
- En este caso sería el poner la mesa
- Usar Kinect para hacerlo más intuitivo
  - Aumentar la motivación

(ventajas herramienta digital frente a una analógica)

Desarrollo de aplicaciones con Microsoft Kinect

Desarrollo de aplicaciones con Microsoft Kinect



Desarrollo de aplicaciones con Microsoft Kinect

- Realizar las acciones usando comandos de voz:
  - Uso de una mano
  - Coger: take
  - Dejar utensilio: leave
  - Comprobar fin: finish
  - Reiniciar: restart
- Problemas:
  - Sólo entiende palabra exacta
  - No debe haber ruido ambiental
  - Tiempo de espera reconocimiento
    - Cambio de icono al detectar hipótesis de palabra

Desarrollo de aplicaciones con Microsoft Kinect

- Uso de dos manos para interactuar
- Coger/dejar utensilios ➡ Manos como pinzas
- Acabar juego ➡ Manos sobre botón fin
- Reiniciar juego ➡ Manos sobre botón fin
- Problemas:
  - Vibración del esqueleto ➡ Se cogen y dejan objetos sin querer
  - Solución:
    - Hay que separar mucho más las manos del tamaño del Utensilio

Desarrollo de aplicaciones con Microsoft Kinect

- Pruebas de usuario
- Ciertos problemas de interacción
  - Interacción ideal usando los dedos
  - Espacio de juego demasiado grande
- Motivación por parte de los usuarios
- Reducción de tiempos al jugar una segunda vez

Este desarrollo y evaluación forman parte de un *paper* presentado a IDC 2012

Desarrollo de aplicaciones con Microsoft Kinect

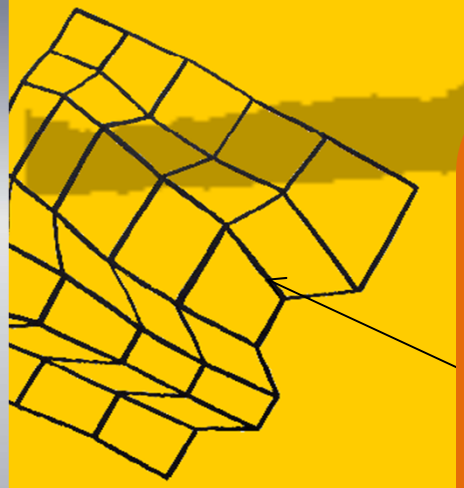
- Isabel Sánchez Gil,
  - licenciada en bellas artes
  - experta en artes visuales y educación
- Exposición 27 abril en el polvorín de la ciudadela

  “Cuando el umbral diferencial no es visible”

  (organizado por el ayuntamiento de Pamplona)
- Una pantalla en la que con ayuda de Kinect
  - Los visitantes puedan “jugar” con obras suyas (y trozos de ella)

Cada visitante pueda plasmar su propia visión

Desarrollo de aplicaciones con Microsoft Kinect



Obra a manejar:
-En cada pantalla, el usuario

la aplicación
nuevo usuario cambia.
rie de colores y
uencialmente

Se considera que un usuario ha abandonado el campo de juego:

-Después de un tiempo sin recibir datos del skeleton de ese usuario (dos segundos)

-Al detectar otro esqueleto, se convertirá en el nuevo usuario

las dos manos
-Se deja cada vez que las manos se cruzan

usuario:
ámara de
d de Kinect
un nuevo usuario
a su silueta
ando aparezca el primer
ujo ésta se desvanecerá
ras un tiempo de inactividad
del usuario, volverá a aparecer

Desarrollo de aplicaciones con Microsoft Kinect

- Reutilización de código
- Selección de acciones basados en experiencia
- Trabajo con alguien ajeno a la informática
- Expuesto al público durante dos meses

Desarrollo de aplicaciones con Microsoft Kinect

- Herramienta nueva
  - Gran interés ➡ Muchos equipos trabajando sobre la herramienta.
  - En continua evolución, con nuevos proyectos.
  - Actualizaciones periódicas de su SDK.
- Prototipo de aplicación educativa
  - Estructura modular ➡ Cambio SDK sin afectar al resto de la aplicación
  - Bajo coste, herramientas accesibles para cualquier hogar
  - Primeros test con usuario
  - Enviado como parte de un paper al congreso (en revisión)
    Interaction Design and Children 2012 (IDC) Bremen (Alemania)
- Aplicación ámbito del arte
  - Expuesta al público a partir del 27 de abril en el polvorín de la Ciudadela

Desarrollo de aplicaciones con Microsoft Kinect

- Experiencia adquirida
  - Nivel técnico
  - Proyecto abierto
  - Trabajo con usuarios finales
- Investigación y desarrollo
  - Investigación y pruebas de diferentes alternativas
  - Experiencia al participar en una publicación

Desarrollo de aplicaciones con Microsoft Kinect

- Desarrollo de la aplicación:
  - Añadirle funcionalidades
  - Hacerlo más específico para un público determinado
  - Englobarlo dentro de un cuaderno de actividades cotidianas

- Mejora de la interacción
  - Probar otros SDK no oficiales
  - Utilizar otras herramientas similares (WAVI Xtion de ASUS)