ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Desarrollo del teclado virtual "Mokey" basado en gestos para personas con movilidad reducida: capa de sistema

AUTOR: Marcos López García

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

TUTOR (o Director en su caso): Martina Eckert

DEPARTAMENTO: Teoría de la Señal y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Mª Luisa Martín Ruiz

VOCAL: Martina Eckert

SECRETARIO: Enrique Rendón Angulo

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN

La Realidad Aumentada forma parte de múltiples proyectos de investigación desde hace varios años. La unión de la información del mundo real y la información digital ofrece un sinfín de posibilidades. Las más conocidas van orientadas a los juegos pero, gracias a ello, también se pueden implementar Interfaces Naturales. En otras palabras, conseguir que el usuario maneje un dispositivo electrónico con sus propias acciones: movimiento corporal, expresiones faciales, etc.

El presente proyecto muestra el desarrollo de la capa de sistema de una Interfaz Natural, Mokey, que permite la simulación de un teclado mediante movimientos corporales del usuario. Con esto, se consigue que cualquier aplicación de un ordenador que requiera el uso de un teclado, pueda ser usada con movimientos corporales, aunque en el momento de su creación no fuese diseñada para ello. La capa de usuario de Mokey es tratada en el proyecto realizado por Carlos Lázaro Basanta.

El principal objetivo de Mokey es facilitar el acceso de una tecnología tan presente en la vida de las personas como es el ordenador a los sectores de la población que tienen alguna discapacidad motora o movilidad reducida. Ya que vivimos en una sociedad tan informatizada, es esencial que, si se quiere hablar de inclusión social, se permita el acceso de la actual tecnología a esta parte de la población y no crear nuevas herramientas exclusivas para ellos, que generarían una situación de discriminación, aunque esta no sea intencionada. Debido a esto, es esencial que el diseño de Mokey sea simple e intuitivo, y al mismo tiempo que esté dotado de la suficiente versatilidad, para que el mayor número de personas discapacitadas puedan encontrar una configuración óptima para ellos.

En el presente documento, tras exponer las motivaciones de este proyecto, se va a hacer un análisis detallado del estado del arte, tanto de la tecnología directamente implicada, como de otros proyectos similares. Se va prestar especial atención a la cámara Microsoft Kinect, ya que es el hardware que permite a Mokey detectar la captación de movimiento. Tras esto, se va a proceder a una explicación detallada de la Interfaz Natural desarrollada. Se va a prestar especial atención a todos aquellos algoritmos que han sido implementados para la detección del movimiento, así como para la simulación del teclado.

Finalmente, se va realizar un análisis exhaustivo del funcionamiento de Mokey con otras aplicaciones. Se va a someter a una batería de pruebas muy amplia que permita determinar su rendimiento en las situaciones más comunes. Del mismo modo, se someterá a otra batería de pruebas destinada a definir su compatibilidad con los diferentes tipos de programas existentes en el mercado. Para una mayor precisión a la hora de analizar los datos, se va a proceder a comparar Mokey con otra herramienta similar, FAAST, pudiendo observar de esta forma las ventajas que tiene una aplicación especialmente pensada para gente discapacitada sobre otra que no tenía este fin.

ABSTRACT

During the last few years, Augmented Reality has been an important part of several research projects, as the combination of the real world and the digital information offers a whole new set of possibilities. Among them, one of the most well-known possibilities are related to games by implementing Natural Interfaces, which main objective is to enable the user to handle an electronic device with their own actions, such as corporal movements, facial expressions…

The present project shows the development of Mokey, a Natural Interface that simulates a keyboard by user's corporal movements. Hence, any application that requires the use of a keyboard can be handled with this Natural Interface, even if the application was not designed in that way at the beginning.

The main objective of Mokey is to simplify the use of the computer for those people that are handicapped or have some kind of reduced mobility. As our society has been almost completely digitalized, this kind of interfaces are essential to avoid social exclusion and discrimination, even when it is not intentional. Thus, some of the most important requirements of Mokey are its simplicity to use, as well as its versatility. In that way, the number of people that can find an optimal configuration for their particular condition will grow exponentially.

After stating the motivations of this project, the present document will provide a detailed state of the art of both the technologies applied and other similar projects, highlighting the Microsoft Kinect camera, as this hardware allows Mokey to detect movements. After that, the document will describe the Natural Interface that has been developed, paying special attention to the algorithms that have been implemented to detect movements and synchronize the keyboard.

Finally, the document will provide an exhaustive analysis of Mokey's functioning with other applications by checking its behavior with a wide set of tests, so as to determine its performance in the most common situations. Likewise, the interface will be checked against another set of tests that will define its compatibility with different softwares that already exist on the market. In order to have better accuracy while analyzing the data, Mokey's interface will be compared with a similar tool, FAAST, so as to highlight the advantages of designing an application that is specially thought for disabled people.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 INTRODUCCIÓN

1.1 Motivación

Este proyecto se desarrolló dentro del Grupo de Realidad Aumentada del CITSEM (Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad). El trabajo ha sido realizado a lo largo de un año y medio como parte de prácticas curriculares en el centro durante el primer medio año, así como con una beca de colaboración durante el año restante. Debido a la complejidad y extensión del programa implementado, se ha desarrollado en conjunto con Carlos Lázaro Basanta, quien analiza detalladamente la capa de usuario de Mokey en [1].

La Realidad Aumentada busca la convergencia de la información que proporciona el mundo real, con la proporcionada por las de las nuevas tecnologías, información virtual. Esta combinación de datos, da como resultado un nuevo mundo de opciones a los desarrolladores, mucho más allá de crear juegos con un mayor grado de interacción. Las Interfaces Naturales son un ejemplo de ello, buscando la comunicación con los dispositivos tecnológicos mediante los movimientos y acciones naturales de una persona.

Según datos de la Organización Mundial de la Salud, en el año 2012 había en el mundo en torno a 1000 millones de personas discapacitadas [2]. Pese a que eso implica 1/7 de la población, la percepción que tiene la sociedad de esta cifra jamás se acercaría a su verdadera dimensión. Vivimos en un siglo en el que lo importante es la sociedad y esta necesita sentir que hace algo por el bienestar del resto, sin embargo, muchas veces, cosas que a priori parecen positivas, no lo son.

Cuando surge la palabra integración en algún proyecto, para prácticamente todo el mundo tiene una connotación positiva. Sin embargo, la totalidad de los proyectos amparados bajo este término, buscan el desarrollo de aplicaciones o dispositivos nuevos, dirigidos únicamente a un grupo que está en riesgo de exclusión social. En otras palabras, se crean pequeños subgrupos dentro de la sociedad para estas personas.

Al crear pequeños subconjuntos de población, se está incurriendo en otro tipo de discriminación, ya que no se está permitiendo el acceso a las cosas usuales por parte de estos sectores de la población. Si se habla de inclusión, se habla de desarrollar aplicaciones y dispositivos que permitan el acceso a las personas en riesgo de exclusión social a las cosas cotidianas. Esto implicaría no crear subgrupos y, por lo tanto, que la sociedad viese en su día a día a las personas discapacitadas. De esta forma, no sólo se observaría lo grande que es el problema, sino que se eliminarían muchas barreras físicas y sociales.

En la actualidad, la tecnología es esencial en las vidas de las personas. Es prácticamente imposible pensar en el día a día sin tener acceso a un ordenador, un Smartphone, Internet, etc. Para una correcta inclusión social, las personas con discapacidad motora o movilidad reducida deberían poder usar cualquier aplicación o juego de un ordenador. Pensando en este sector de población y gracias a la Realidad Aumentada, surge la idea de Mokey (*Motion Keyboard*), una Interfaz Natural que permite el uso de cualquier programa de ordenador mediante movimientos del usuario.

Mokey surge con el objetivo de romper la barrera que existe para el acceso a los diferentes programas y juegos de ordenador por parte de las personas con discapacidades motoras o movilidad reducida. Por todo esto, los objetivos han de verse desde un punto de vista mucho más allá del tecnológico, buscando una solución que sea real y funcional para este segmento de la población.

1.2 Objetivos

El objetivo principal es el desarrollo de una Interfaz Natural que permita el uso de cualquier aplicación de un ordenador mediante movimientos. Está pensada tanto para gente discapacitada como para la posible realización de ejercicios por lo que, en base a esto, se han de estructurar los objetivos del proyecto.

El objetivo principal es el desarrollo de un middleware que conecte la información de movimiento que proporciona un dispositivo de captura de movimiento, utilizando concretamente la cámara Microsoft Kinect [3], con los eventos de teclado que necesitan las aplicaciones. Al estar pensada para personas con movilidad reducida, se tienen que tener en cuenta los siguientes sub-objetivos:

- Crear un banco de movimientos representativos que sea lo más útil posible para los diferentes tipos de discapacidades motoras.
- Crear parámetros de ajuste que permitan adecuarse a las necesidades de cada aplicación y usuario.
- Crear una interfaz de usuario simple e intuitiva. Esto cobra especial importancia al estar hablando de un sector de la población que no tiene fácil acceso al uso de un ordenador, por lo que, para muchos, puede resultar complicado su manejo.
- Crear una función que permita su uso en silla de ruedas. Gran parte de la gente con una discapacidad motora, usa silla de ruedas.
- Crear un módulo que permita al usuario la grabación de movimientos propios. De esta forma, podrá adecuarse más a las necesidades personales de cada usuario.
- Conseguir una optimización del middleware para que utilice los mínimos recursos posibles del ordenador. Eso es esencial para que la aplicación que se quiere controlar con movimientos no vea afectado su rendimiento.

1.3 Estructura

La presente memoria se estructura en un total de cinco capítulos y dos anexos. En el primer capítulo, se proporciona al lector una visión general de las motivaciones y objetivos del proyecto.

En el segundo capítulo se analiza detalladamente la tecnología empleada para el desarrollo de Mokey. Esto proporciona al lector los conocimientos necesarios para poder seguir con facilidad los capítulos siguientes. También se realiza un estudio del arte de los programas similares que han sido desarrollados con anterioridad.

En el tercer capítulo se presenta el desarrollo de la solución implementada. Se presta especial atención en la explicación conceptual de los algoritmos implementados. Debido a la extensión del código, en este capítulo el lector no va a encontrar muchas referencias

al mismo, en caso de desear más información sobre la utilización de estos algoritmos en el lenguaje de programación, se ha elaborado el Anexo 2.

En el cuarto capítulo se realiza un análisis exhaustivo del rendimiento de Mokey y de su compatibilidad con otras aplicaciones. Para ello se somete a dos baterías de pruebas que intentan abarcar las situaciones más usuales en las que se podría usar la interfaz.

El capítulo final ofrece al usuario una visión general de todo lo visto en los capítulos anteriores. Finalmente se ofrece una visión del futuro a medio plazo de Mokey, indicando las posibles mejoras que se analizará si introducir.

El Anexo 1 proporciona al usuario la información necesaria para la instalación y manejo del programa.

Capítulo 2 ANTECEDENTES

El primer paso antes de desarrollar una solución a un problema, es estudiar en profundidad las tecnologías implicadas en dicha solución, así como aquellas que son afines porque su objetivo es similar. En este capítulo se va a realizar un estudio profundo de estas dos áreas tecnológicas.

2.1 Software y Hardware empleados en Mokey

En este apartado se explican con detalle todos los dispositivos y software que van a ser necesarios para el desarrollo de Mokey. Se va a prestar especial atención a las versiones usadas por el propio proyecto, en los casos en los que se hayan producido actualizaciones recientes.

2.1.1 Cámara Microsoft Kinect

En el año 2010 Microsoft lanzó al mercado la cámara Microsoft Kinect para la videoconsola Xbox 360 [4]. Este dispositivo revolucionó la forma en la que las personas podrían interactuar con los videojuegos, permitiendo el manejo de los mismos mediante el movimiento corporal, sin la necesidad de que el usuario sostenga dispositivos hardware para fijar su posición, haciendo la experiencia más real. En la figura 2.1 se puede observar el aspecto de la cámara Microsoft Kinect.

Fig. 2.1 Microsoft Kinect para Xbox 360

La cámara Microsoft Kinect tiene tres funciones principales: detección del movimiento corporal del usuario, detección de los gestos faciales del usuario y reconocimiento de voz. Estas funciones fueron, en principio, diseñadas para manejar la Xbox 360, pero no pasó mucho tiempo hasta que los desarrolladores vieron el potencial que tenía la cámara para y querían aprovechar la tecnología para otras aplicaciones en ordenadores.

A finales del año 2010 la fundación OpenNI [5] empezó a proporcionar drivers que permitían el uso de la Kinect en un ordenador, abriendo un nuevo mundo de posibilidades tecnológicas. En cuanto Microsoft fue consciente del potencial que tenía, no solo proporcionó su propia SDK para poder usar la Kinect en dispositivos Windows, sino que, además, sacó al mercado en el año 2012 una nueva cámara especialmente pensada para ello, *Kinect for Windows*.

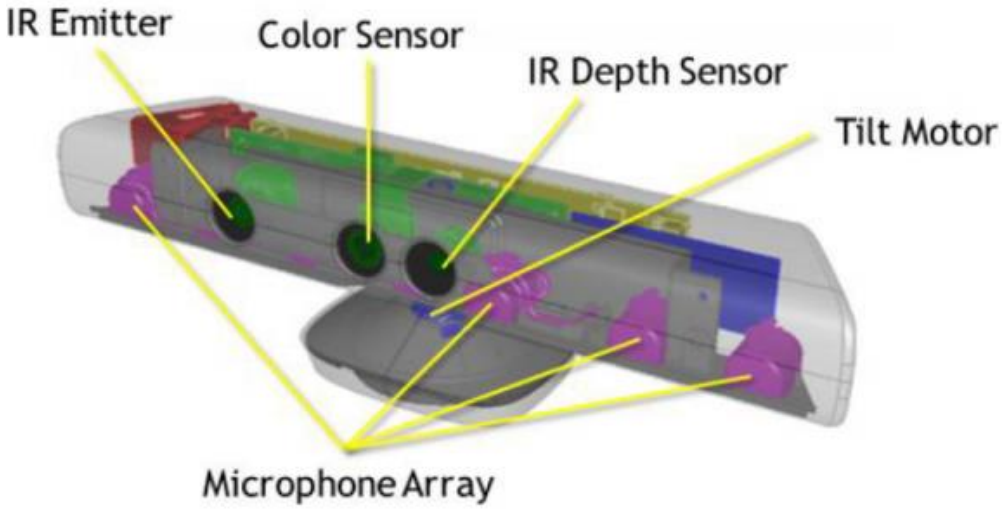

Fig. 2.2 Elementos de la Microsoft Kinect v1

En la figura 2.2 [6] se observan los principales elementos de la Kinect. En primer lugar, se puede observar una cámara RGB, que aparece denominada como *Color Sensor*. Esta cámara actúa como lo haría cualquier cámara convencional, con una resolución que se puede ver en la tabla 2.1. Cuenta con dos dispositivos infrarrojos, un emisor, *IR Emitter*, y un receptor, *IR Depth Sensor*. Gracias a estos dos elementos, se consigue generar la imagen de profundidad. El motor, denominado *Tilt Motor*, permite la orientación de la cámara en el plano vertical, adaptándose así a las necesidades derivadas de la altura a la que se encuentre la Kinect. Finalmente, el array de micrófonos, denominado *Microphone Array*, es el encargado de la captación de audio.

Los rayos infrarrojos rebotan contra los objetos que se encuentran en su camino. Gracias a esto, los rayos emitidos que reboten contra el usuario serán posteriormente recibidos por el receptor, que, como sabe en qué instante de tiempo han sido emitidos, puede determinar cuánto camino han recorrido y, por extensión, su distancia. En la figura 2.3 se pueden observar la imagen RGB y la imagen de profundidad con las que trabaja la Kinect.

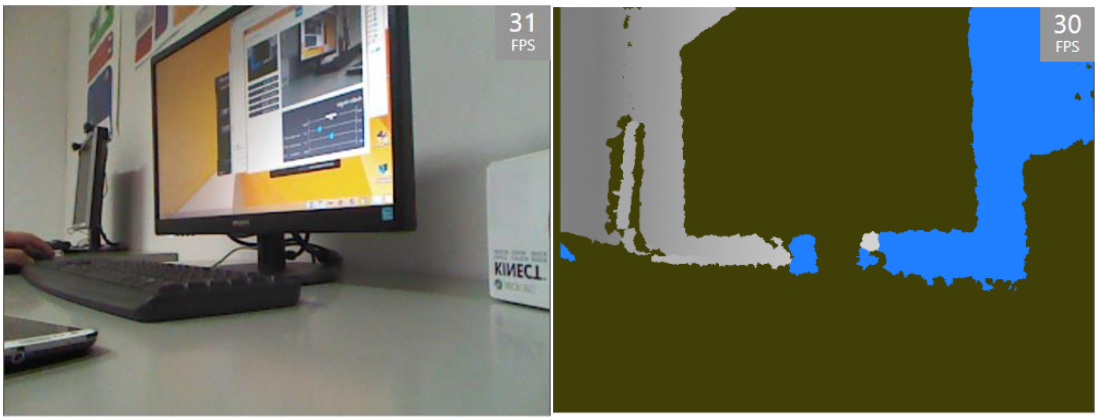

Fig. 2.3 a) Imagen RGB de la Kinect b) Imagen de profundidad de la Kinect

Una vez determinados los principios de funcionamiento de la Kinect, es el momento de profundizar más en bajo qué condiciones ha de usarse. Gracias a las imágenes RGB y de

profundidad, la SDK, kit de desarrollo de software, genera el esqueleto del movimiento, pero, para que pueda hacerlo, se tienen que cumplir unos requisitos. La cámara no es capaz de detectar a más de dos usuarios a la vez, por lo que una tercera persona impediría el correcto funcionamiento de la misma. La distancia a la que se encuentra el usuario es esencial; teóricamente, la Kinect es capaz de detectarlo si se encuentra entre 1,2 m y 3,5 m. En la práctica, este rango no es tan amplio. En cuanto al ángulo de apertura horizontal y vertical de la cámara, estos son 57° y 43°, respectivamente.

En lo referente a la captación de movimiento corporal, la SDK de Kinect es capaz de determinar hasta 20 articulaciones del usuario, aportando sus tres coordenadas espaciales (X,Y,Z) y su rotación. Esto será ampliamente tratado en el capítulo 3.

Algunas de las limitaciones y problemas que tiene la Kinect vienen marcados por la propia tecnología utilizada. Ya se ha visto el rango de captura de la cámara, si el usuario se encuentra fuera de este, se va a producir una simulación de movimiento de la parte del usuario que no es registrada por la cámara. Como cualquier predicción, puede distar mucho de lo que está sucediendo realmente y, consecuentemente, generar falsas detecciones. Del mismo modo, cuando alguna de las articulaciones es tapada por otra parte del cuerpo al realizar un movimiento, a esta no están llegando los rayos infrarrojos y, de nuevo, se ha de recurrir a la simulación, siendo en este caso mucho más inexacta y generando movimientos del esqueleto rápidos e incoherentes.

Como ya se ha visto antes, hay dos versiones de la Kinect 1, una para la Xbox 360 y otra especialmente pensada para Windows. Pese a esta diferenciación, ambas son perfectamente compatibles con Windows. La segunda es posterior en el tiempo y, por lo tanto, presenta una serie de mejoras, sin alterar en ningún caso los fundamentos tecnológicos y estéticos de la misma.

Las principales mejoras de la *Kinect for Windows* fueron orientadas a los sensores. Estos son más sensibles y mejoran sustancialmente el tiempo de ejecución de las acciones derivadas de ellos. Otro de los cambios más llamativos es su rango de visión, contando en esta ocasión con dos modos, modo por defecto y modo cercano. En el primero de ellos, que es el equivalente al comentado en la versión para Xbox 360, el rango aumenta ligeramente situándose entre los 0,8 m y los 4 m. No hay cambios en los ángulos de captura. En cuanto al modo cercado, permite captar entre los 0,4 m y los 3 m. Aunque la introducción de este modo parece dotar a la Kinect de nuevas posibilidades, lo cierto es que Microsoft no garantiza grandes cosas al usarlo; no garantiza cuantos puntos del esqueleto podrán ser detectados y condiciona su correcto funcionamiento a la iluminación.

Cuando se está hablando del diseño de aplicaciones, el flujo de datos y la calidad de los mismos son esenciales. En función del proyecto, puede ser más importante una cosa o la otra. *Kinect for Windows* mejora ampliamente este aspecto, como se puede observar en la Tabla 2.1.

Como se puede ver en dicha tabla, las sustanciales mejoras que proporciona la versión para Windows, van sobretodo enfocadas a poder trabajar con mayores resoluciones y, por extensión, con mayor calidad.

Tabla 2.1 Comparativa de resoluciones de la Kinect para Xbox y para Windows

	Kinect for Xbox 360	Kinect for Windows
Img. Profundidad	320 x 240 a 16 bits @ 30fps	80 x 60 a 16 bits @ 30fps 320 x 240 a 16 bits @ 30fps 640 x 480 a 16 bits @ 30fps
Img. Color	640 x 480 a 32 bits @ 30fps	640 x 480 a 32 bits @ 30fps 1280 x 960 RGB @12fps Raw YUV 640 x 480 @15fps
Audio	Audio de 16 bits @ 16 kHz	Audio de 32 y 64 bits

Por último, cabe destacar que también se implementó una gran mejora en la detección de audio, permitiendo el reconocimiento de un mayor número de lenguas. También es interesante que, en un mismo ordenador, se pueden usar hasta 4 *Kinect for Windows* de forma simultánea, lo cual no es posible con la versión de la Xbox 360.

Tras esta comparativa, hay que indicar que, aunque para este proyecto la *Kinect for Windows* hubiese sido la indicada, únicamente se dispone para la realización del mismo de la versión para la Xbox 360. Sin embargo, ya que comparten SDK y el programa es independiente, es perfectamente compatible con la versión para Windows y, simplemente con usar esta, se obtendrían las mejoras comentadas, sin realizar ningún cambio en el programa.

A finales del año 2013, fue presentada la versión 2 de la Kinect [7]. En la figura 2.4 se muestra esta nueva versión. Las principales novedades respecto a su predecesora son la captación de hasta 6 usuarios, la mejora de la resolución hasta los 1080p y, por extensión, una mayor precisión a la hora de detectar los movimientos corporales, así como la detección de los gestos faciales.

Fig. 2.4 Aspecto de la Microsoft Kinect v2

Uno de los datos más importantes de la Kinect 2 es que, a diferencia de en su anterior versión, Microsoft únicamente ha lanzado una versión al mercado. Esto implica que, para poder usar la cámara en soportes con Windows, hay que utilizar un adaptador. Otro dato importante es que ya han sido liberadas las herramientas para desarrolladores, sin embargo, Microsoft es muy celosa con la información que proporciona y, a día de hoy, la

información disponible no es equiparable a la que hay de la versión 1, limitando las oportunidades de los desarrolladores.

2.1.2 SDK Microsoft Kinect v1.8

La SDK de Microsoft Kinect incluye el conjunto de drivers, aplicaciones, herramientas y funciones que permiten a los desarrolladores programar aplicaciones utilizando como soporte su cámara. La versión aquí analizada es la última actualización para la Kinect 1, que fue lanzada en septiembre de 2013[8].

Es importante señalar que esta versión de la SDK es compatible con todas las predecesoras, así como con Windows 8.1 y versiones anteriores. La SDK permite desarrollar aplicaciones en dos lenguajes de programación: C# y C++. La versión actualizada incluye todas las herramientas de las anteriores. A continuación, se van a explicar algunas de las funciones más destacadas:

- **Body tracking**: proporciona las herramientas necesarias para definir el esqueleto con el movimiento, gestionar la información que proporciona, dibujar el esqueleto y otras más. Respecto a esta función, la SDK ha ido presentando múltiples utilidades para hacerla más fluida y apta para cada situación.
- **Face tracking**: proporciona las herramientas necesarias para definir la malla que permite identificar los gestos faciales, gestionar esta información y representarla.
- **Detección de voz**: proporciona las herramientas necesarias para gestionar el audio captado por los micrófonos, situarlo espacialmente y usarlo para detección de voz e interpretación de texto.
- **Otras herramientas**: proporciona otra serie de herramientas menos conocidas que permiten, entre otras cosas, combinar personas con fondos de su elección, es decir, realizar una especie de croma, pero usando la información de la profundidad para detectar al usuario. Basándose en este principio, presenta otras herramientas que permiten la sustitución de, por ejemplo, la cabeza del usuario por otra o poner la suya en algún personaje.

En el momento de escribir este apartado, ya está en el mercado la SDK Microsoft Kinect v2. Sin embargo, esta versión es para la Kinect 2, por lo que no es de utilidad para el desarrollo de Mokey.

2.1.3 Lenguaje de programación C#

C# es un lenguaje de programación orientado a objetos diseñado por Microsoft. Hizo su primera aparición en el año 2000 y, posteriormente, sería estandarizado en la norma ISO/IEC 23270 [9]. Es un lenguaje que tiene influencias de C, C++ y Java.

Pese al largo recorrido de este lenguaje de programación, su penetración en el mercado no es muy alta. En el ranking de lenguajes de programación más utilizados en el año 2014 por Gartner, C# está situado en el puesto número 8, por detrás de algunos lenguajes como Java, C, Python, PHP o C++. A pesar de esto, Microsoft apuesta por él y, por ello, lo introduce en muchas de las SDK de sus dispositivos más famosos, como la Kinect.

A pesar de estos esfuerzos por parte de Microsoft por aumentar su popularidad, lo cierto es que no proporciona grandes ventajas respecto a otros lenguajes similares, como Java o

C++ y, sin embargo, sí tiene grandes inconvenientes. Las librerías proporcionadas por Microsoft son bastante básicas y presentan algunos errores o carencias que hace años que han sido subsanados en sus competidores.

Por otro lado, la comunidad de programadores no es muy activa en los foros con este lenguaje. Esto hace que, ante posibles problemas e incidencias que se encuentre el programador, sea muy difícil encontrar una solución por canales oficiales o no oficiales.

2.1.4 Visual Studio 2013

Visual Studio 2013 es un entorno de desarrollo de aplicaciones de Microsoft que permite la edición, depuración y compilación de programas en diferentes lenguajes [10]. Visual Studio es una de las herramientas de programación más versátiles del mercado, ofreciendo 7 versiones de su producto que se adaptan a las necesidades de cada programador. Por orden de complejidad son:

- Visual Studio 2013 Ultimate
- Visual Studio 2013 Premium
- Visual Studio 2013 Test Pro
- Visual Studio 2013 Pro
- Visual Studio Online Básico
- Visual Studio Online Pro
- Visual Studio Online Avanzado

La versión utilizada para el proyecto desarrollado en esta memoria es Visual Studio 2013 Ultimate, es decir, la más versátil de todas y la que contiene todos los módulos posibles. Esto implica que algunas de las características aquí vistas no aparezcan en versiones inferiores.

Los lenguajes de programación que incorpora Visual Studio de forma predefinida son: C#, C++, F#, JavaScript, Basic y HTML. Algunos lenguajes como C# o F# han sido desarrollados por Microsoft, por lo que, aunque no tienen una penetración de mercado tan amplia como Java o C++, gozan de un gran desarrollo en esta herramienta al ser también de su invención.

Es muy importante destacar que Visual Studio permite muchos más lenguajes además de estos cinco, sobre todo en lo referente a lenguajes derivados de estos, pero, en este caso, sin ofrecer tantos automatismos, que es una de las principales características de este programa, como se verá a continuación.

Visual Studio fue creado con el objetivo de facilitar a los programadores el desarrollo de aplicaciones y hacer el proceso lo más automático e intuitivo posible. En otras palabras, busca sustituir, en la medida de lo posible, el escribir líneas de código, por procesos más visuales, en los que el usuario tenga que seleccionar y arrastrar elementos. En la figura 2.5 se puede observar la primera ventana que muestra Visual Studio a la hora de crear el proyecto. En ella queda patente esta idea; tras seleccionar uno de los múltiples lenguajes de los que se dispone, el programa creará automáticamente todas las funciones, referenciará las librerías más comunes e implementará una estructura básica. Esta funcionalidad puede ahorrar al programador varias horas de programación.

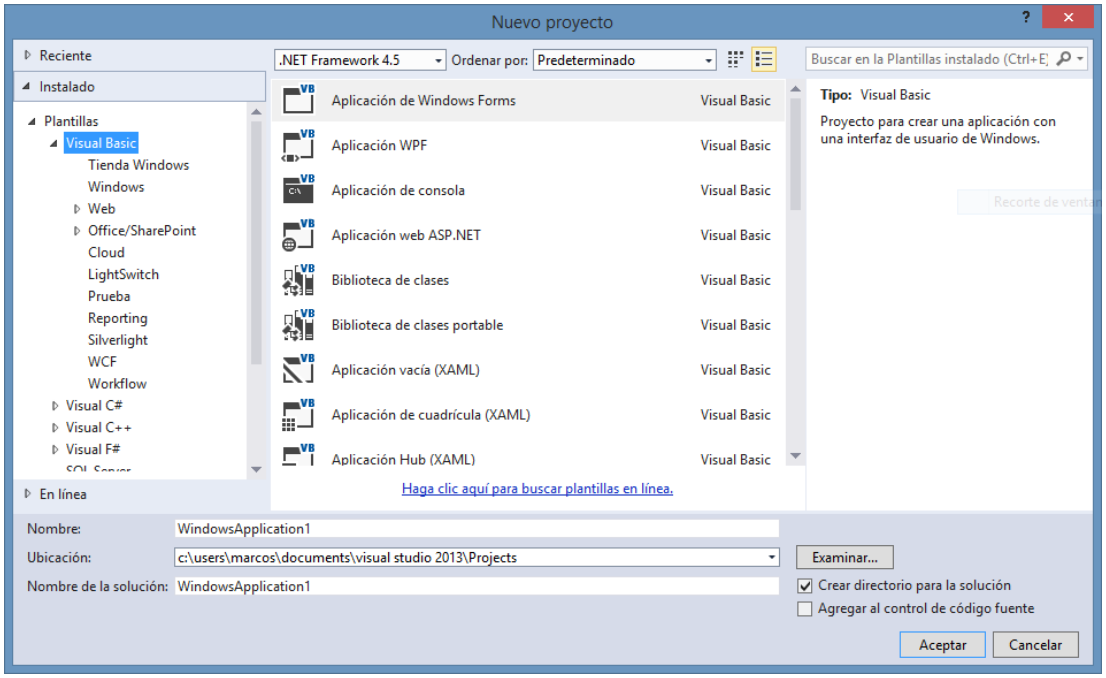

Fig. 2.5 Ventana de creación de proyecto en Visual Studio

Continuando con esta línea de hacer un proceso de programación más visual e intuitivo que de escribir código, Microsoft desarrolla su característica más conocida: crear una interfaz gráfica totalmente funcional sin la necesidad de crear código. Para ello, Visual Studio proporciona los dos módulos que se pueden observar en las figuras 2.6 y 2.7, respectivamente.

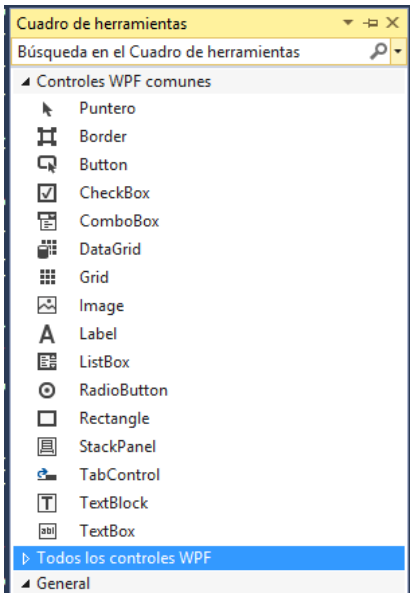

Fig. 2.6 Módulo de selección de elementos

Para la generación de la interfaz, hay que elegir alguno de los elementos de la lista desplegable que se muestra en la figura 2.6. En esta lista se pueden elegir botones, barras deslizadores, listas, cajas de texto, etc. Para su utilización, simplemente hay que arrastrar el elemento seleccionado al entorno gráfico que se observa en la figura 2.7. Una vez

soltado dicho elemento en la interfaz, podrá ser desplazado manualmente a gusto del programador.

En la parte inferior de la imagen 2.7 se observa un código en formato xaml, es decir, un lenguaje de programación diferente al del programa diseñado. Este código se genera automáticamente en función de la creación visual que el programador está realizando. Es importante destacar que el usuario puede programar también directamente sobre ese código, siendo esto especialmente útil para colocar cada elemento a la distancia indicada y que todo quede simétrico.

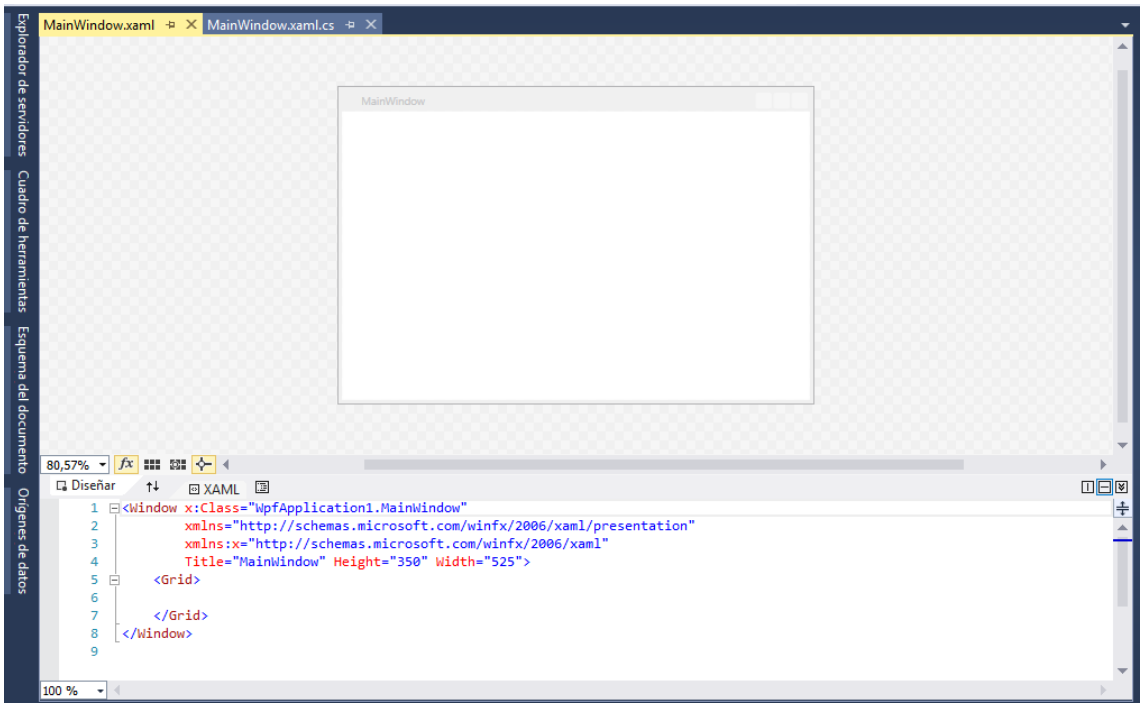

Fig. 2.7 Representación visual de la interfaz gráfica.

Llegados a este punto, y como se puede ver en la figura 2.7, cada programa cuenta con dos archivos de código diferente, uno en xaml para la interfaz gráfica y otro con el programa en el lenguaje designado. La comunicación entre ambos archivos, se lleva a cabo mediante los denominados eventos que ha desarrollado Microsoft para dicho propósito.

Los eventos son una serie de mensajes, que son capaces de comunicar cada uno de los elementos de la interfaz gráfica con una función en el programa principal, que ha sido especialmente diseñada para ello. La creación de estas funciones es totalmente automática, y simplemente hay que hacer doble clic sobre el elemento en el archivo xaml para que esta se cree en el archivo principal, ya en el lenguaje de programación que se esté utilizando. Es importante destacar que cada elemento puede tener diferentes eventos posibles. Por defecto, al hacer doble clic se activa el más usual; para usar el resto se ha de ir al menú dedicado a ello.

Visual Studio, al igual que otros editores-compiladores como Eclipse, permite la compilación del texto y su posterior ejecución desde el propio programa. Del mismo modo, permite situar puntos de ruptura en el propio código, que da acceso al valor de las diferentes variables del programa en ese punto del código.

Visual Studio cuenta con más módulos que amplían su versatilidad hasta hacerlo apto casi para las necesidades de cualquier programación, sin embargo, no se van a tratar por no verse implicados en el desarrollo de este proyecto.

Recientemente, Microsoft ha anunciado la próxima salida al mercado de Visual Studio 2015, que parece tener como objetivo ampliar el número de lenguajes que son empleados y hacer más fácil la programación de aplicaciones para Windows Phone.

2.2 Estado del Arte

En este apartado se presenta un estudio de las tecnologías similares a Mokey que hay actualmente en el mercado o están en desarrollo. De entre todos los programas que usan la Kinect para el diseño de una Interfaz Natural, hay uno de especial interés en este proyecto por su similitud con la solución presentada: FAAST. Por ello, se va a prestar especial atención a dicho programa, dedicándole un apartado propio en el que se analizará detalladamente. Posteriormente, se proporcionará una visión general del resto de estudios y aplicaciones.

2.2.1 FAAST

The Flexible Action And Articulated Skeleton Tookit, más conocido como FAAST, es una Interfaz Natural diseñada por South California University que permite simular eventos de teclado en un ordenador mediante movimientos corporales [11].

FAAST utiliza la Kinect como dispositivo de captación de movimiento, permitiendo el uso de OpenNI o la SDK de Microsoft Kinect como controladores de ella. Esto lo dota de una mayor versatilidad en cuanto a que la eficiencia de las dos librerías puede variar y, en función de lo que esté buscando el usuario, una de ellas puede ser más útil que la otra.

Es una Interfaz Natural que está pensada para ofrecer al usuario la mayor versatilidad posible. Para ello, se pueden generar tantos movimientos como sean necesarios y asociarlos a los eventos de teclado correspondientes.

En la figura 2.9 se puede observar la interfaz gráfica de FAAST. En ella, se puede ver que está compuesta de un total de 3 módulos. En el primero de ellos, arriba a la izquierda, se encuentran todos los elementos que permiten la configuración de FAAST. A su derecha, se observa una ventana negra que muestra, cuando la Kinect está activa, la imagen de profundidad y el esqueleto que genera. Por último, en la parte inferior de la figura, se observa la consola de FAAST, en la que se pueden ver todas las acciones que van ejecutándose en el programa.

Permite también la simulación del cursor del ratón, lo que lo dota de una gran capacidad para ser compatible con multitud de aplicaciones y juegos de ordenador.

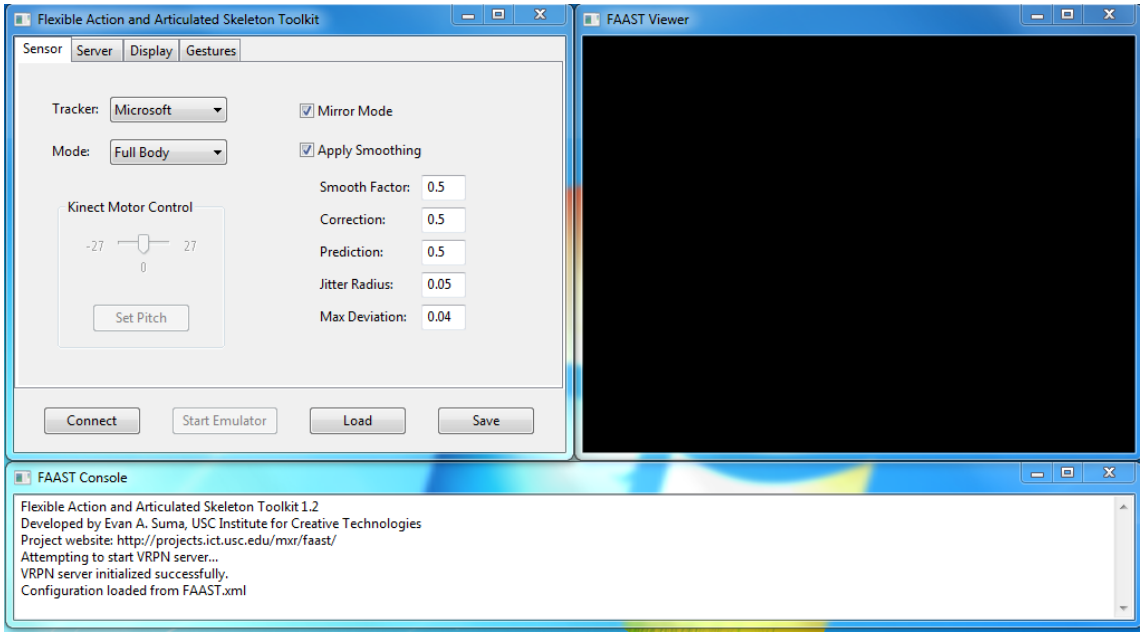

Fig. 2.8 Interfaz gráfica de FAAST

En la figura 2.9 se puede observar la ventana en la que se crean los movimientos. Como se puede ver, el usuario tiene que determinar manualmente todos los datos de interés. Esto implica que el jugador ha de tener un conocimiento sobre las articulaciones con las que cuenta el esqueleto generado por la Kinect. Del mismo modo, ha de tener en cuenta las distancias mínimas existentes entre las dos partes seleccionadas, es decir, la cadera va a estar separada del hombro en el eje x, por una distancia mínima siempre. FAAST no aporta ningún tipo de limitación ni control a la hora de determinar si los movimientos configurados son realizables. Esto, puede generar una sucesión reiterada de eventos de teclado que el usuario no quería producir.

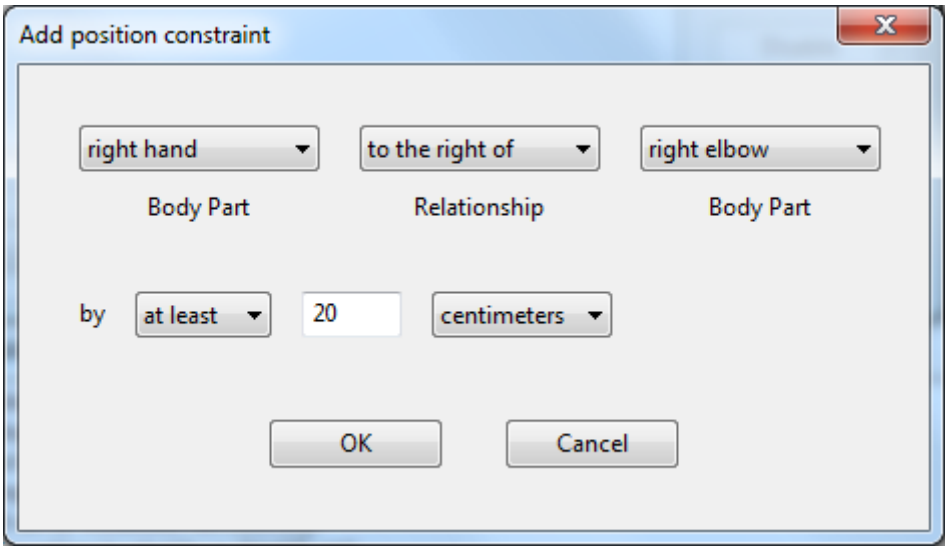

Fig. 2.9 Módulo de configuración de movimientos de FAAST

Los desarrolladores de FAAST han comunicado que la última versión, lanzada en diciembre de 2013, es la última actualización que se realizará sobre el software.

2.2.2 Otras tecnologías

FAAST ha sido el punto de partida de muchos estudios que pretenden mejorarlo o complementarlo. En [12] se presenta un entorno de trabajo, en el cual se puedan ejecutar las aplicaciones sobre las que se quiere usar la Interfaz Natural, minimizando los problemas típicos de compatibilidad. Este tipo de investigaciones buscan solventar la brecha que existe entre el diseño de las aplicaciones cotidianas y las Interfaces Naturales que buscan sustituir el teclado y el ratón por acciones naturales. En muchas ocasiones, la forma en la que están programadas dificulta que estas puedan ser usadas por estas Interfaces.

Otros autores han desarrollado Interfaces que buscan sustituir los eventos de teclado o ratón por comando de voz o gestos con las manos, como se puede ver en [13]. Este tipo de líneas de investigación, fomentan el acceso al software usual por cualquier persona, sea cual sea su problema. Los tipos de discapacidad y sus grados son muy amplios, por lo que, en muchos casos, hay que intentar buscar la unificación de diferentes modos de interacción con las aplicaciones para que cubran todas las necesidades.

En el momento en el que la Kinect salió al mercado, múltiples centros de investigación y universidades vieron el potencial que tenía. En el periodo temporal que abarca del año 2011 al 2012, se pueden encontrar múltiples estudios que tratan el seguimiento corporal con la Kinect y, en otros muchos, la simulación de un ratón gracias a estos movimientos. Algunos ejemplos de esto último son KinEmote, PCD o WIN&I. La mayoría de desarrollos de este tipo no han sido mantenidos en el tiempo, dejando de proporcionar soporte o actualización.

Como se ha visto en el apartado 2.1.1, la Kinect fue pensada como un medio para controlar la Xbox 360. Los desarrolladores fueron los que propiciaron el uso de la cámara en programas para Windows y Unix, con el desarrollo de librerías como OpenNI. Desde este momento y hasta que la SDK de Microsoft fue lanzada, fueron múltiples los estudios que surgieron sobre el estudio del tracking corporal, es decir, el seguimiento del movimiento corporal. Algunos de estos estudios son [14], [15], [16] o [17].

La mayoría de estas publicaciones basan su estudio en la librería OpenNi y tratan, con mayor o menor profundidad, el diseño de sistemas que permiten el seguimiento de las personas. Una de las cosas más interesantes de este punto de inflexión en la investigación de este campo es que, en algunos de los estudios como [17], se presenta un desarrollo matemático muy detallado que permite entender de dónde vienen las funciones que, a día de hoy, todos los investigadores utilizan sin necesidad de realizar estos cálculos de forma consciente, ya que son zonas opacas de las funciones.

Tras la salida de la SDK proporcionada por Microsoft, todos estos estudios se ven interrumpidos, ya que la librería proporcionada por ellos cumplía con las necesidades que todos los desarrolladores tenían. Este es otro nuevo punto de inflexión, en el cual los proyectos sobre OpenNi empiezan a desaparecer para dar lugar a una nueva generación basada en la SDK.

Otra de las líneas de investigación surgida a raíz de la Kinect, fue la de utilizar los gestos de las manos para controlar el ordenador o incluso para interpretar el lenguaje de signos. Esto se puede ver por ejemplo en [18]. Bajo determinadas circunstancias y determinado

filtrado, se ha logrado interpretar algunos gestos simples con las manos. Sin embargo, la resolución de la Kinect impide interpretar la alta variedad de movimientos existentes en la lengua de signos, no haciéndola apta para este propósito.

Por último, hay que hablar de la capacidad de la Kinect para sistemas orientados a la salud. En [19] se muestra una introducción de cómo tiene capacidad para ser utilizada en este campo. Ahora mismo se está asistiendo a un crecimiento de los estudios que usan la Kinect para rehabilitación y su uso médico en quirófanos. Todo parece indicar que, gracias a la nueva Kinect, esto va a seguir produciéndose a corto plazo.

Capítulo 3 DESCRIPCIÓN TÉCNICA DE LA INTERFAZ NATURAL

A lo largo de este capítulo se van a explicar detalladamente todos los aspectos técnicos de la Interfaz Natural Mokey. Se enfoca principalmente a los algoritmos usados y los recursos utilizados, sin profundizar en el código de programación. Se incluye un anexo con la información de mayor valor, así como comentarios sobre el propio código del programa.

Los detalles referentes a cómo Mokey interactúa con los usuarios, son tratados en [1]. A lo largo del capítulo, se indicarán aquellos aspectos que se deben buscar en dicho proyecto. La apariencia de la interfaz gráfica de Mokey se puede observar en la figura 3.1.

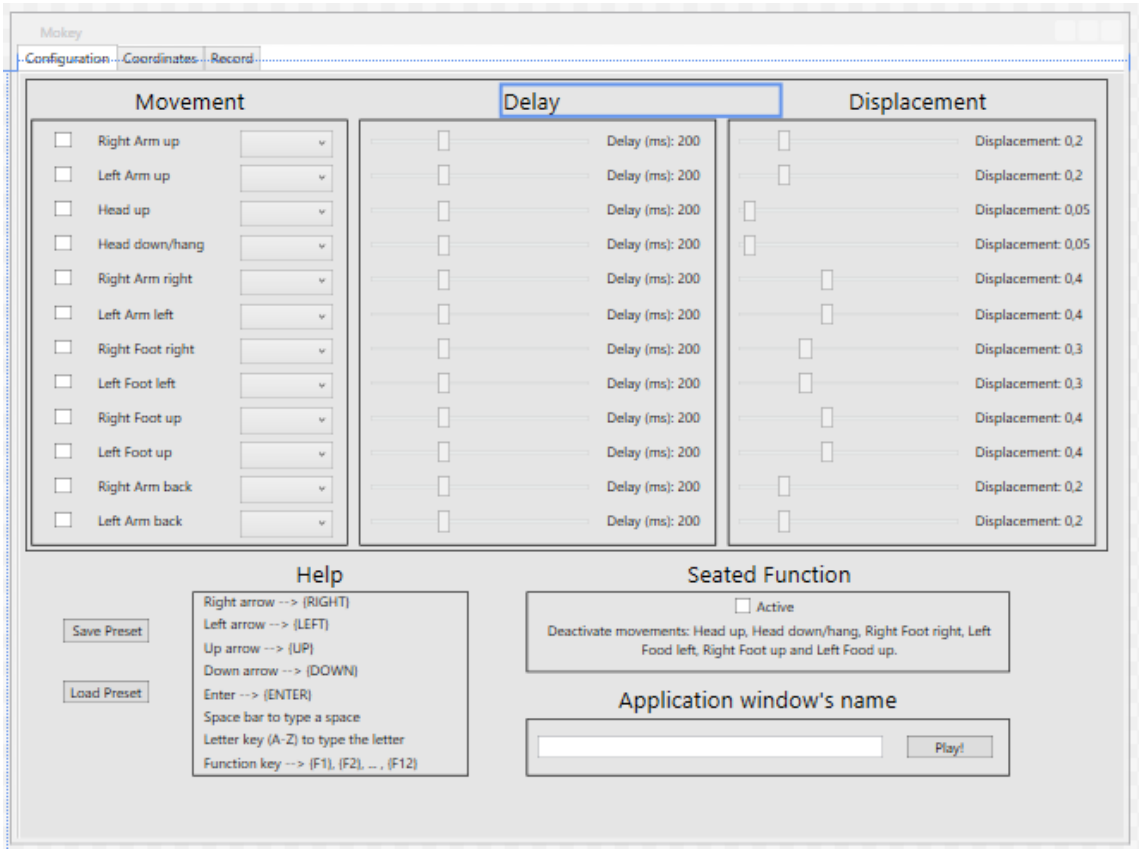

Fig. 3.1 Interfaz gráfica de Mokey

En la figura 3.1, se puede observar que la interfaz está dividida en un total de 6 módulos. Tres se corresponden a la configuración de los movimientos, permitiendo elegir la tecla asignada a cada caso, el retardo entre pulsaciones y la amplitud del movimiento. En la esquina inferior izquierda, se observa el módulo que permite guardar y cargar las configuraciones de usuario. Para usarlo, únicamente hay que pulsar dichos botones y se generará o solicitará el archivo con la configuración. El quinto módulo hace referencia a la función sentado. Finalmente, el módulo en el que se introduce el nombre de la ventana en primer plano y tras el cual, al pulsar sobre "Play!", se activarán todos los hilos generados. . Debido a la complejidad de implementación de la misma y la importancia

que tiene para su interacción con el usuario, esta parte está ampliamente desarrollada en [1].

3.1 Evolución del proyecto

Como se ha visto con anterioridad, el proyecto tiene dos etapas temporales. La primera con una duración de 6 meses, durante el transcurso de prácticas curriculares. Durante este periodo se elabora la conexión de la Kinect con Mokey, así como el teclado virtual. Los movimientos a elegir son muy reducidos, ya que la prioridad en este momento es conseguir conexiones estables entre las diferentes partes implicadas. El resultado final es una versión del programa que permite al usuario elegir hasta 6 movimientos y asignarles una tecla. También permite elegir parámetros como su retardo o desplazamiento, que serán analizados más adelante.

La segunda etapa del proyecto es posible gracias a una beca de colaboración del Ministerio de Educación Cultura y Deporte. Durante este periodo, se busca dotar a Mokey de mayor versatilidad y simpleza, haciéndolo acto para las personas con movilidad reducida o discapacidad motora. Las características finales son las siguientes:

- Mokey permite usar hasta 12 movimientos predefinidos, de los que se puede regular su amplitud, es decir, cuán amplio ha de ser el desplazamiento para que sea válido.
- Mokey permite generar eventos de teclado con variaciones de pulsación de entre 0 ms a 600 ms.
- Cada movimiento configurado se ejecuta en un proceso independiente y paralelo.
- Mokey permite grabar hasta 4 movimientos con los brazos y las piernas.
- Mokey puede ser usado por personas en silla de ruedas. En este caso pueden usar 6 de los movimientos predefinidos y podrán ampliarlos con los 4 que se pueden grabar.
- Mokey permite seleccionar la aplicación sobre la que se quiera usar, para que aparezca en primer plano cuando la Kinect detecte que la persona ya está delante.
- Mokey permite guardar la configuración para futuros usos.
- Mokey permite la modificación de todos los parámetros en tiempo real.

3.2 Descripción de los algoritmos empleados

En el apartado anterior se ha observado que hay cuatro procesos esenciales en Mokey: la configuración, la extracción de las coordenadas, la comparación de las coordenadas con los criterios y, finalmente, la generación de eventos de teclado. En esta sección, se van a explicar detalladamente los algoritmos implementados para el correcto funcionamiento de cada una de estas fases.

3.2.1 Extracción de las coordenadas del esqueleto

Gracias a la SDK de Kinect [8], se pueden obtener las coordenadas espaciales de los 20 puntos que se observan en la figura 3.2 [20]. Estos puntos están definidos por las coordenadas cartesianas (X, Y, Z) y su rotación.

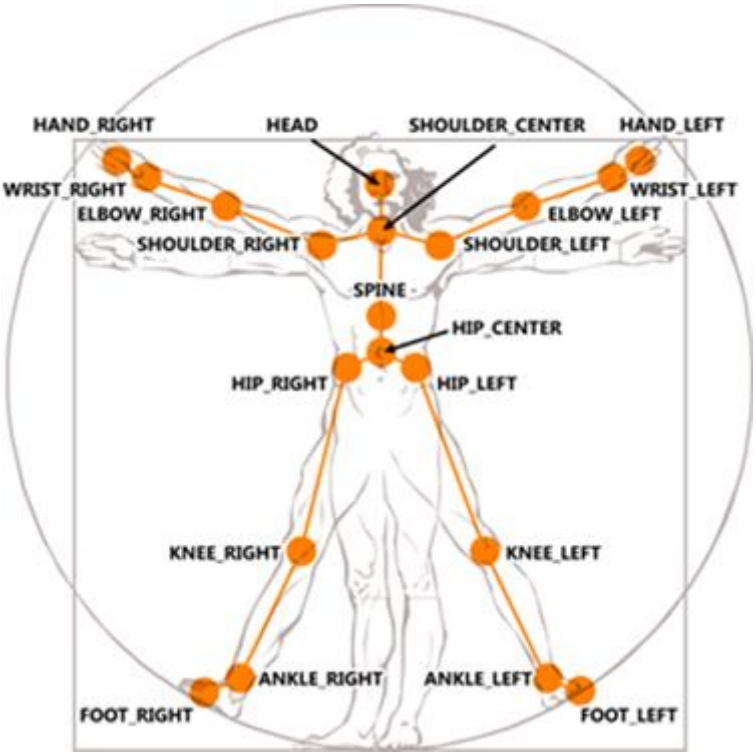

Fig. 3.2 Representación de los puntos que proporciona la Kinect

En este apartado solo se va a tratar la parte directamente implicada en la extracción de las coordenadas por su relevancia. Para mayor información sobre la parte del código implicada en la definición de los huesos y la Kinect, se ha desarrollado el anexo 2.

El flujo de datos generado por el esqueleto puede llegar a ser bastante alto. Debido a esto, es esencial utilizar solo la información imprescindible, de forma que el middleware utilice los menores recursos posibles de CPU y RAM.

Para garantizar este uso moderado de recursos, los 12 movimientos que es capaz de detectar Mokey de forma predeterminada solo se basan en la información de 6 de los veinte puntos. Los 6 puntos de los que se obtienen sus coordenadas son: *Foot_right*, *Foot_left*, *Hand_right*, *Hand_left*, *Spine* y *Head*.

Una vez se ha limitado la cantidad de puntos, hay que entender cómo procesar de forma eficiente la información que se obtiene de dichos puntos. Para facilitar al máximo los cálculos, únicamente se van a utilizar las coordenadas cartesianas (X,Y,Z). Ahora falta entender respecto a qué se miden dichas coordenadas. La coordenada Z hace referencia a la distancia en metros entre la Kinect y la persona. Para el caso de la X y la Y, se aprovecha como referencia el dibujo del esqueleto que nos permite la SDK en una ventana. Para clarificarlo, en la figura 3.3 a) se puede observar la representación del esqueleto que se puede realizar con la SDK, en la figura 3.3 b) se observa sobre esta representación una simulación de la escala de referencia que usa internamente la SDK para situar el esqueleto.

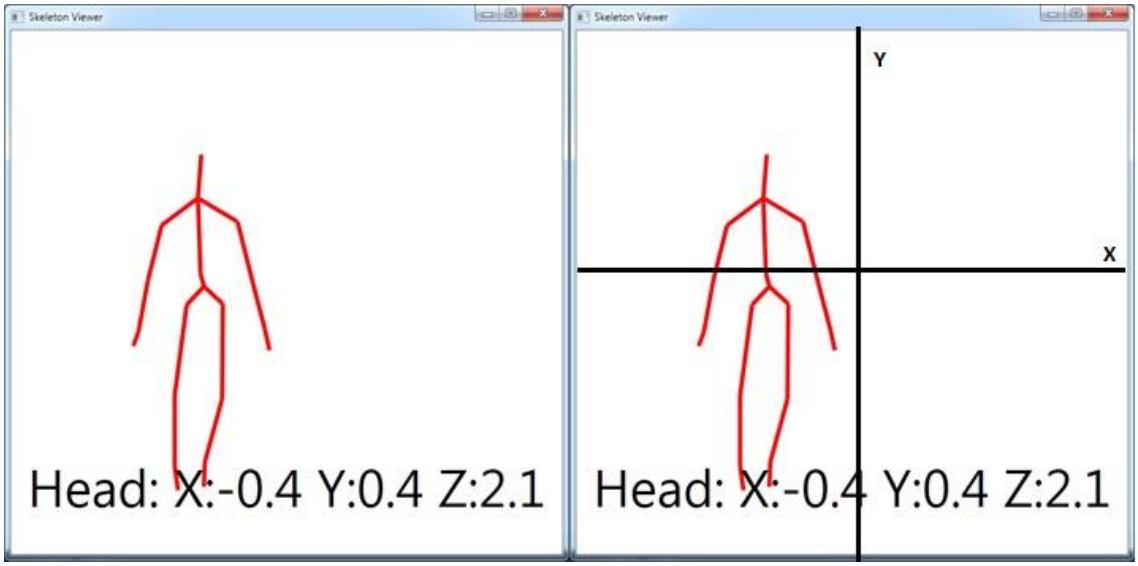

Fig. 3.3 a) Esqueleto dibujado por la SDK de Kinect b) esqueleto dibujado por la SDK de Kinect con los ejes de la escala superpuesta

Se puede observar en la figura 3.3 b) que el punto de la cabeza se encuentra situado en la zona negativa del eje x, y en la positiva del eje y. En cuanto al eje z, sirve para determinar que, en este caso, el usuario estaba a 2.1 metros de la Kinect. Los ejes X e Y también tienen en este caso unidades en metros respecto al eje de referencia.

Finalmente, una vez definidos los huesos o puntos que se van a usar y cómo se van a tratar sus coordenadas, se necesita extraer dicha información. Para ello, se va a utilizar una utilidad de la SDK que permite escribir en un *String* las coordenadas de uno de los puntos. En la siguiente línea de código se muestra la realización de dicha extracción para la mano derecha:

```
message = string.Format("RightArm: X:{0:0.0} Y:{1:0.0} Z:{2:0.0}",
                ini_brazoDer_X = handRigthPosition.X,
                ini_brazoDer_Y = handRigthPosition.Y,
                        handRigthPosition.Z);
```

En la variable *message*, que es un *string*, se guarda el texto que aparece como primer parámetro. El segundo, tercero y cuarto, sirven para actualizar las coordenadas (X,Y,Z) que se observan en el primer parámetro. Los números 0, 1 y 2 que se encuentran en primera posición entre las llaves {}, indican en qué variable deben de mirar para actualizarse. A la vez que se actualiza este mensaje, que es el que se muestra en la pestaña coordenadas de la interfaz gráfica, se aprovecha para guardar esas coordenadas en tres variables. Ini_brazoDer_X, Ini_brazoDer_Y y Ini_brazoDer_Z, son tres variables globales de tipo float, que almacenan temporalmente la posición de las coordenadas de interés para que sean accesibles desde todas las funciones. Es importante que este proceso se realice en un bucle constante en el tiempo, para obtener toda variación de movimiento realizada por el usuario.

Tras estos pasos, se están actualizando constantemente las coordenadas de los puntos de interés, que, a su vez, se pasan a una variable global para ser accesible en cualquier función del programa.

3.2.2 Uso de las coordenadas para la detección del movimiento

Gracias a las coordenadas obtenidas como se ha visto en el apartado anterior, se pueden implementar los 12 movimientos que se observan en la tabla 3.1. Dichos movimientos son exhaustivamente analizados en [1]. En este apartado, se va a desarrollar la explicación del algoritmo empleado en el código para determinar si dichos movimientos cumplen o nos los criterios asignados.

Tabla 3.1 Relación de movimientos y huesos implicados

Movimiento	**Hueso principal**	**Hueso de referencia**	**Eje de interés**
Brazo derecho arriba	*Right Hand*	*Spine*	y
Brazo derecho a la derecha	*Right Hand*	*Spine*	x
Brazo derecho atrás	*Right Hand*	*Spine*	z
Brazo izquierdo arriba	*Left Hand*	*Spine*	y
Brazo izquierdo a la izquierda	*Left Hand*	*Spine*	x
Brazo izquierdo atrás	*Left Hand*	*Spine*	z
Pierna derecha arriba	*Right Foot*	*Spine*	z
Pierna derecha a la derecha	*Right Foot*	*Spine*	x
Pierna izquierda arriba	*Left Foot*	*Spine*	z
Pierna izquierda a la izquierda	*Left Foot*	*Spine*	x
Bajar cabeza/ agacharse	*Head*	No se aplica	Y con calibración
Levantar cabeza / puntillas	*Head*	No se aplica	Y con calibración

Mokey es un middleware, por lo tanto, como ya se ha nombrado y realizado en pasos anteriores, hay que reducir en la medida de lo posible su gasto computacional. Debido a esto, se ha tomado la decisión de que cada movimiento esté definido únicamente por una de las tres coordenadas, desechando las otras dos. En la tabla 3.1 queda indicado cuál es el eje usado en cada movimiento. Este hecho, además, facilita el uso del programa por personas que no pueden realizar movimientos estables, como se puede ver en [1].

Otro dato a destacar de la tabla 3.1, es que los 10 primeros movimientos tienen asociado un hueso de referencia. El algoritmo calcula la distancia existente en el eje de interés, entre ambos huesos. Esto facilita el uso del programa ya que, independientemente de la posición del usuario, las distancias relativas entre huesos no varían. Sí cabe destacar que podrían provocarse algunos casos de hipersensibilidad en la detección de movimientos si el usuario está muy lejos de la Kinect.

Una vez explicados los dos puntos esenciales de la detección del movimiento, se va a proceder a explicar cada uno en detalle. En primer lugar en la figura 3.4, se puede observar la posición estática del usuario, que será en base a la cual se compararán todos los desplazamientos.

Fig. 3.4 Posición estática del usuario

En la figura 3.5 se observa el dibujo correspondiente al movimiento del brazo derecho hacia la derecha.

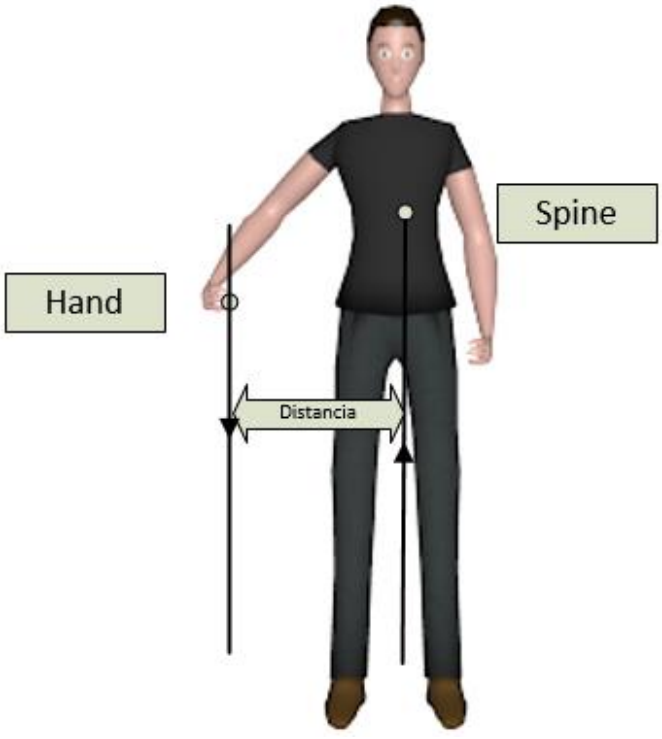

Fig. 3.5 Representación del movimiento del brazo derecho hacia la derecha

Como se puede observar en la figura 3.5, la distancia en el eje x entre el hueso de la columna (*Spine*) y el de la mano derecha (*Hand*), aumenta al separar el brazo. Gracias a este incremento de distancia al realizar este movimiento, se puede comprobar el cumplimiento del mismo. Este mismo esquema es válido para el razonamiento del movimiento del brazo izquierdo a la izquierda. A continuación, en la figura 3.6, se observa el caso del brazo derecho hacia arriba.

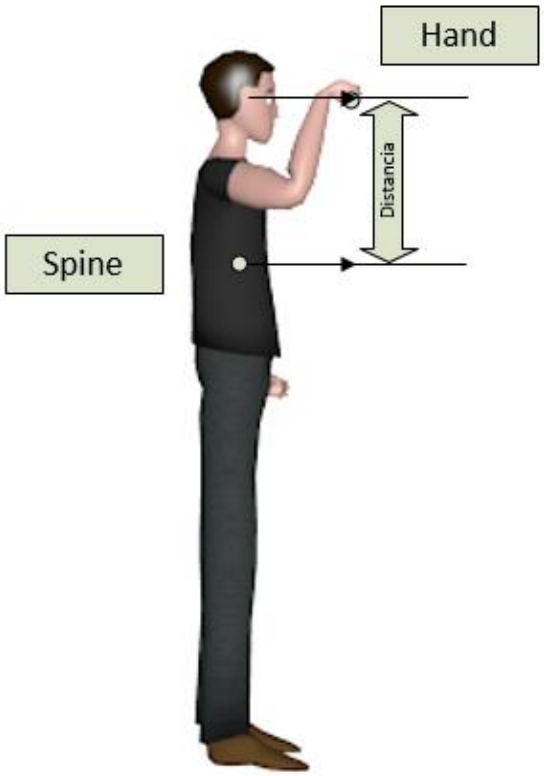

Fig. 3.6 Representación del movimiento del brazo derecho hacia arriba

En el caso de la figura 3.6, se observa que la distancia que aumenta es la existente entre el eje y de la columna y la mano. Hay que destacar algo muy importante: la mano está en su posición de reposo por debajo del hueso *spine*. Esto genera un aumento de la distancia absoluta no lineal ya que, en los primeros instantes de subir el brazo, la distancia disminuye en vez de aumentar. Sin embargo, esto no afecta al resultado puesto que, por lo general, es bastante fácil superar el umbral de la columna. Este esquema también es representativo de lo que sucede en el caso del movimiento del brazo izquierdo hacia arriba.

En la Figura 3.7 se muestra otro caso, cuando la pierna derecha se mueve a la derecha.

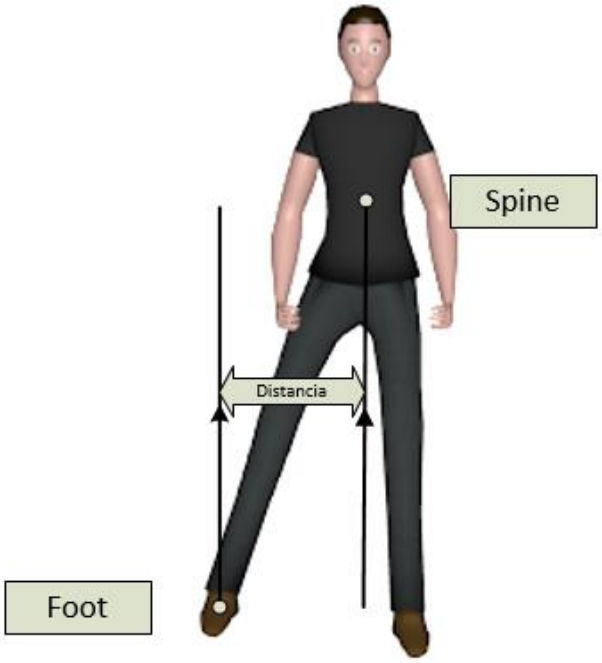

Fig. 3.7 Representación del movimiento de la pierna derecha hacia la derecha

En el caso mostrado en la figura 3.7, se compara la distancia en el eje x que existe entre la columna y el pie derecho que siempre es positivo. Si el usuario cruzase las piernas, el valor de esta distancia en la variable que la almacena sería negativo debido al orden de la resta, por lo que no se cumpliría el criterio La misma manera de detección es aplicable al movimiento de pierna izquierda a la izquierda.

En la figura 3.8 se observa el movimiento de la pierna derecha hacia delante.

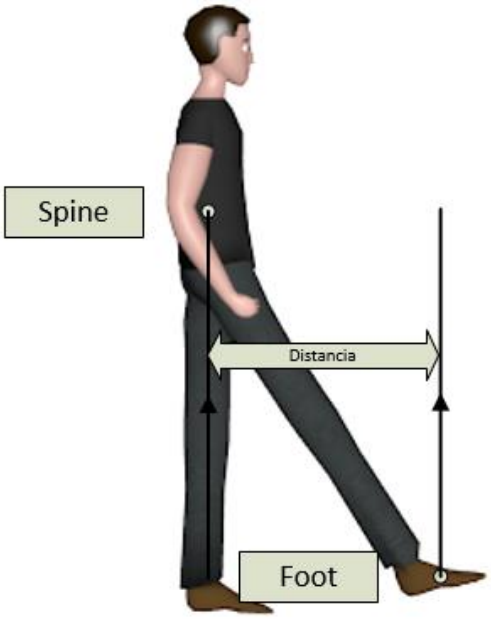

Fig. 3.8 Representación del movimiento de la pierna derecha hacia delante

En la figura 3.8, se observa cómo al mover la pierna derecha hacia delante. Esto también es representativo en el caso del movimiento de la pierna izquierda hacia adelante. En la figura 3.9, se observa la representación de los dos últimos movimientos que no necesitan calibración: mover los brazos hacia atrás.

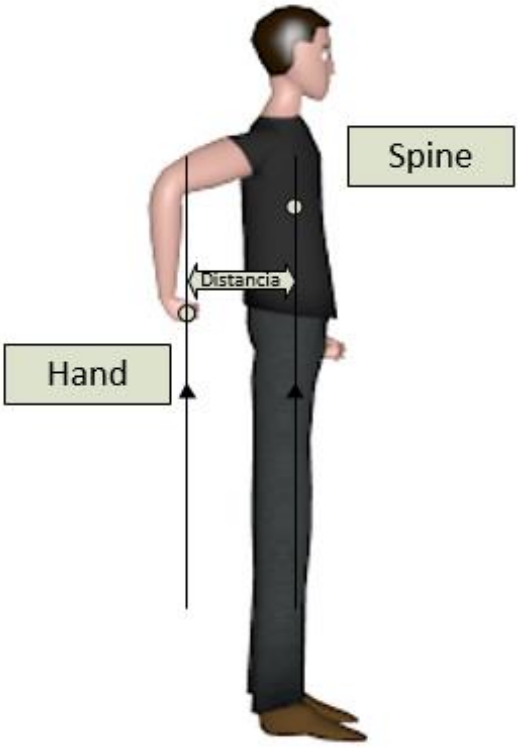

Fig. 3.9 Representación del movimiento del brazo derecho hacia atrás

En la figura 3.9, se observa cómo la distancia en el eje z, entre la mano y la columna, aumenta al desplazar el brazo hacia atrás. Es muy importante destacar que este movimiento puede causar problemas con la detección si el brazo es tapado por el torso. En esta situación, la simulación del esqueleto empieza a ser errática, produciendo incoherencias en las coordenadas.

Que los movimientos solo se determinen con la variación en uno de los ejes, tiene una consecuencia directa, y es que dos de los criterios pueden ser cumplidos para un mismo movimiento. Por ejemplo, el brazo derecho movido hacia arriba en diagonal, cumpliría los requisitos de brazo derecho a la derecha y brazo derecho arriba. Para solucionar esta situación, en aquellos movimientos en los que esto puede darse con facilidad, se han impuesto prioridades, de forma que la posición que, a juicio de los desarrolladores, sea la más probable de las dos, es la correcta. Por ejemplo, volviendo al caso de brazo en diagonal arriba, mover el brazo hacia arriba requiere de un esfuerzo importante, por lo que se consideraría que el usuario está pretendiendo hacer este movimiento y se descartaría el de brazo derecho a la derecha.

Todas estas condiciones, se comprueban mediante condiciones Se compara en cada instante de tiempo la distancia existente entre los puntos de interés al hueso *spine*. El número de veces que se comprueba la posición por segundo varía del retardo asignado al movimiento como se ve más adelante. Si se detecta una distancia mayor que el umbral

definido, se simula el evento de teclado correspondiente. Que el movimiento se produzca si es mayor, facilita el proceso a nivel de programación, pero, la razón principal tiene que ver con el grupo de personas al que va dirigida la interfaz; esto se analiza detalladamente en [1].

3.2.3 Calibración inicial

Como se indica en el apartado anterior, dos de los movimientos configurables necesitan calibración inicial. Esto quiere decir que, para comprobar si el movimiento se cumple, los datos actuales son comparados con otros captados con anterioridad.

Los dos movimientos que necesitan calibración son: agacharse y ponerse de puntillas. Al pulsar el usuario el botón "Play!", si estos movimientos están activos, se le indica que en 5 segundos se realizará la calibración. Esta consiste en guardar, en una variable global, la altura de la persona en este instante. La calibración tiene una consecuencia bastante clara, y es que el jugador no podrá moverse en exceso de la posición en la que se encuentra o, se empezarán a producir falsas detecciones. En la figura 3.10 se observa cómo funciona dicha calibración.

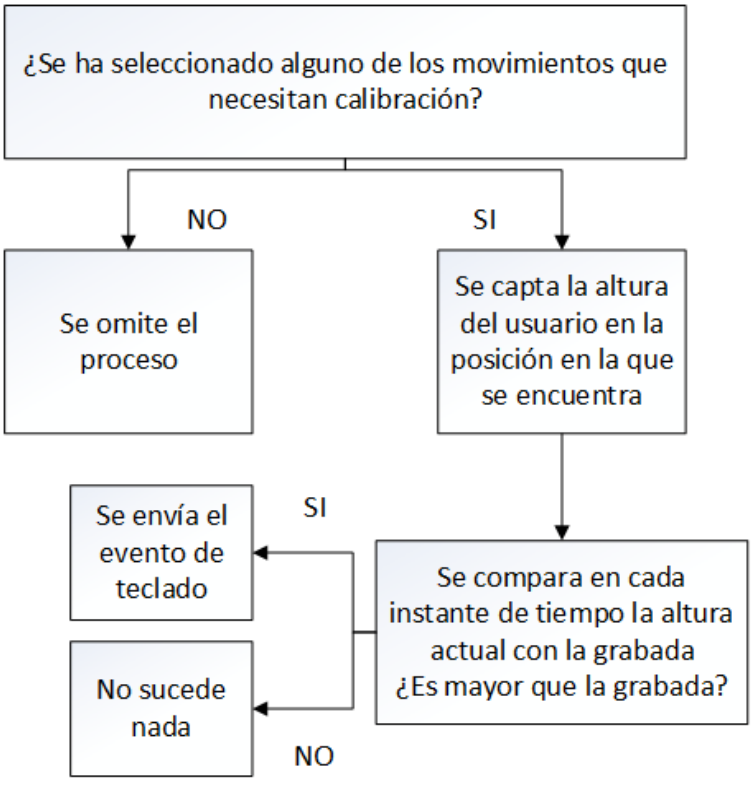

Fig. 3.10 Representación de la calibración

Estos dos movimientos fueron los primeros en ser implementados. Posteriormente, se desarrolló una batería suficientemente grande, como para poder prescindir de estos dos debido a su imprecisión. Esta imprecisión es debida a que el cumplimiento o no de las condiciones de movimiento, dependen de que el usuario esté siempre fijo en la misma posición del espacio, lo que es bastante difícil mientras se está jugando.

3.2.4 Simulación de eventos de teclado

El objetivo de Mokey es generar un evento de teclado, ante la realización de un movimiento captado por la Kinect. Para la simulación de dichos eventos, se emplea la función `Sendkeys()` de la librería de Microsoft *System.Windows.Forms* [21].

Para simular el evento, hay que introducir como parámetro el código correspondiente a la tecla deseada. Esto se hace internamente, el usuario únicamente ha de seleccionar en la lista desplegable de la interfaz gráfica la tecla que quiere usar. En la Tabla 3.2 se pueden observar algunos de los códigos más comunes. Por ejemplo, en caso de querer simular la pulsación de la flecha derecha, el código a escribir es:

```
Sendkeys({RIGHT});
```

Tabla 3.2 Relación de letras y parámetros a introducir

Tecla deseada	Parámetro a introducir literalmente
Cualquier letra o número (Por ejemplo la letra A)	A
Flecha derecha	{RIGHT}
Flecha izquierda	{LEFT}
Flecha hacia abajo	{DOWN}
Flecha hacia arriba	{UP}
Intro	{ENTER}
Espacio	
Teclas de función (Por ejemplo F2)	{F2}

Pese a que, en otros lenguajes de programación, hay funciones similares muy desarrolladas y complejas, en el caso del lenguaje C#, sus posibilidades son muy limitadas. La función no permite simular un evento prolongado en el tiempo, es decir, mantener una tecla pulsada. Únicamente permite generar un evento aislado en el tiempo, es decir, pulsar y soltar la tecla en el acto. Esto, aunque a priori no parezca un gran inconveniente, ya que siempre se puede poner la función en un bucle para simular la pulsación continuada, impide que la función sea válida para algunos juegos antiguos que realmente requieren un pulso continuado y no una cadena de pulsos.

3.2.5 Retardos entre pulsaciones

Uno de los objetivos de Mokey es adaptarse al mayor número de personas y programas posibles. Para esto, hay que considerar que cada programa puede necesitar un número de pulsaciones diferentes en un lapso de tiempo. El retardo define el tiempo que va a trascurrir entre dos eventos de teclado consecutivos, mientras el usuario mantiene el movimiento. Por ejemplo, si se juega al Tetris y el usuario quiere mover una pieza a la derecha, interesa que la separación entre pulsaciones sea grande para que tenga un control preciso sobre cuantos espacios quiere desplazarse. De esta manera con un retardo de 300 ms, el usuario se desplazaría unos tres espacios a la derecha casa segundo que mantenga el movimiento. Por el contrario, si es un juego de carreras, en el que la pulsación implica la aceleración del coche, se necesita que el retardo sea lo más pequeño posible para que el coche no se frene. Si este fuese se 0 ms, se simularía una pulsación continuada en el espacio, aunque esta no fuese del todo real, ya que emite un tren de pulsos consecutivos

y no un único pulso. Del mismo modo, ante determinados tipos de problemas motrices, puede haber usuarios que necesiten regular el espaciado entre pulsaciones para que no se produzcan más de las que desea. Este puede ser el caso de personas con dificultades de control sobre su brazo tras una mastectomía, al tardar más de lo habitual en devolver el brazo a su posición inicial, si el retardo fuese muy amplio, se enviaría más de un evento lo que dificultaría la experiencia de juego. Este último punto, y su programación en la interfaz gráfica, son tratados ampliamente en [1]. En este apartado, se va a explicar cómo se han aplicado los retardos en el código.

Para la implementación de los retardos se usa la función `Thread()` de la librería *System.Object* de Microsoft [22]. La función crea un hilo temporal, una vez transcurrido el tiempo que se le introduce como parámetro (en milisegundos). Este hilo nuevo, no dispone de una rutina de trabajo, pero pausa la ejecución del hilo principal hasta que deje de ser la línea de código ejecutada tras los milisegundos fijados. Por ejemplo, si se desea introducir un retardo de medio segundo, el código es el siguiente:

```
Thread.Sleep (500);
```

Es importante destacar que, pese a crear un hilo temporal para aplicar el retardo, esto no incrementa el gasto computacional, porque dicho hilo no tiene que realizar ninguna operación de CPU y, además, durante el tiempo que dura el retardo, el código principal no se está ejecutando, por lo que no hay ningún conflicto posible en el uso de la CPU.

El usuario puede seleccionar, mediante un slider en la interfaz gráfica de Mokey, el retardo que quiere aplicar a cada movimiento. Podrá modificar los valores entre 0 ms y 600 ms. Este valor máximo ha sido definido basándose en que retardos mayores ya dificultarían el uso normal de aplicaciones, ya que la mayoría requieren de acciones rápidas. El valor con el que se inicia por defecto es de 200 ms. Ha sido elegido porque equivale a cinco pulsaciones por segundo, lo que es suficiente para tener una pulsación ágil y un control suficiente para un usuario sin problemas de movimiento. En la figura 3.11 se puede observar destacado dicho elemento en la interfaz gráfica.

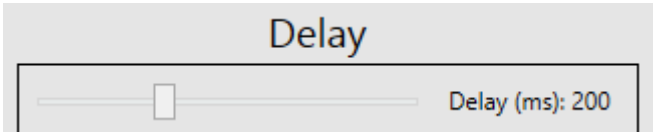

Fig. 3.11 Representación del slider que permite seleccionar el retardo

Es importante destacar que estos retardos han sido empleados para reducir significativamente el gasto computacional. Dado que Mokey compara los puntos actuales de la Kinect para varios movimientos, el número de operaciones por segundo sería extremadamente alto si no se controla de alguna forma. En la figura 3.12 se observan dos casos: a) Gasto elevado de CPU b) Gasto optimizado de CPU.

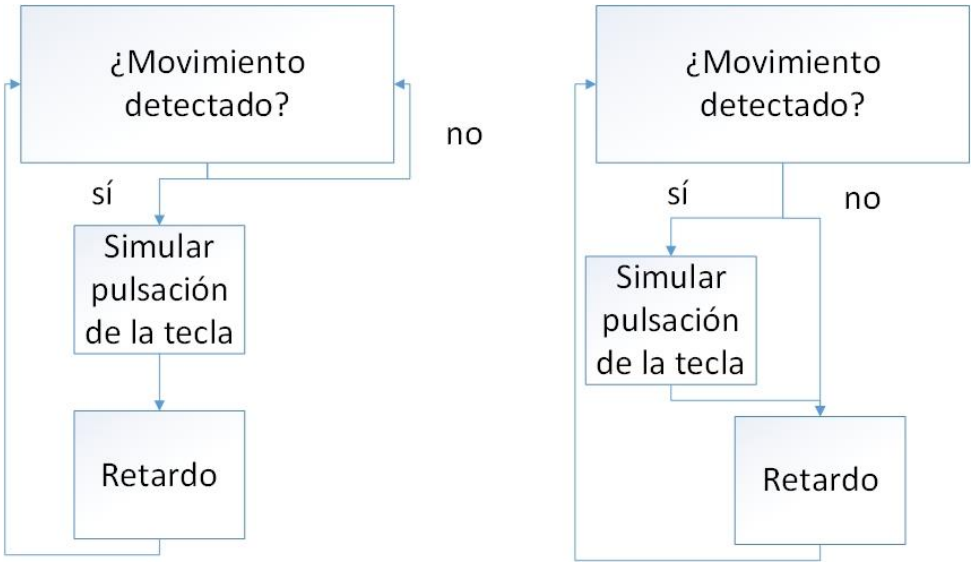

Fig. 3.12 a) Retardo dentro de la condición de movimiento. b) Retardo fuera de la condición de movimiento

En el caso a), se aplica el retardo cuando ya se ha cumplido la condición, es decir, solo se aplica cuando se sabe que ya se ha producido el evento y se quiere garantizar que no se produce otro en los milisegundos indicados. Debido a que no hay nada que limite cuántas veces se realiza la operación, y el código se reproduce en un bucle infinito, ocuparía el 100% de la CPU. En caso de tener una CPU más potente, simplemente se harían aún más operaciones por segundo, pero no quedaría parte libre. En el caso b), se aplica dentro de la rutina de programa donde se comprueba si se cumple o no un movimiento. Se cumpla o no la condición de movimiento, se va a producir el retardo indicado. De esta forma, se pierde algo de resolución, ya que se comprueba menos veces por segundo si se está produciendo el movimiento. Pese a esto, el usuario no va a notar ninguna diferencia entre el caso a) y el b), pero, a cambio, el gasto computacional se reduce significativamente.

Para clarificar la anterior explicación, se va analiza un ejemplo partiendo de las siguientes premisas:

- La CPU de la que se dispone tiene una capacidad máxima de procesado de 100 operaciones por segundo.
- Cada comprobación de movimiento supone una operación.
- El retardo aplicado es de 200 ms.
- El tiempo que se tarda en hacer la operación para comprobar si el movimiento se cumple, es despreciable, es decir se considera de 0 ms.
- El código no tiene herramientas que limiten cuanta CPU puede ser usada por él.

Se procede a calcular el porcentaje de CPU usado en cada uno de los casos, así como las operaciones realizadas en dos casos en un periodo de 1 segundo. En el primer caso no se produce ninguna condición de movimiento, es decir, aunque el usuario puede moverse, ningún movimiento coincide con los configurados. En el segundo se produce una condición de movimiento justo en el instante inicial, es decir, el usuario comienza la prueba con un movimiento que coincide con los configurados, sin realizar posteriormente ninguno más en el segundo restante.

Caso 1. Sin condiciones de movimientos producidas

a) Retardo dentro de la condición

En este caso, el retardo está dentro de la condición, es decir, solo se produce si ésta se cumple. Debido a esto, se harían en ese segundo las 100 operaciones permitidas, es decir, se estaría usando el 100% de la CPU. Esto se debe a que si se observa la figura 3.12 a), se ve que cuando la condición no se cumple, automáticamente vuelve a hacer otra más, sin ningún tipo de limitación.

b) Retardo fuera de la condición

En este caso, el retardo se realiza después de cada cálculo de movimiento, es decir, se cumpla o no la condición. Debido a esto, se haría 1 operación cada 200 ms. Esto da un total de 5 operaciones y un 5% de CPU usado. Esto se debe a que si se observa la figura 3.12 b), se ve que independientemente de que se cumpla o no la condición, antes de realizar la siguiente operación se produce un retardo.

Caso 2. Condición de movimiento producida en instante inicial

a) Retardo dentro de la condición

En este caso, el retardo está dentro de la condición, es decir, solo se produce si ésta se cumple. Se hace la operación inicial y se produce el retardo de 200 ms, pero, tras estos, se vuelven a producir cálculos sin retardos, es decir, otras 80 operaciones. Esto da un total de 81 operaciones y un 81% de CPU usado. Esto se debe a que si se observa la figura 3.12 a), En el instante inicial entra por la rama del sí, produciéndose un retardo de 200 ms tras la primera operación. Tras este periodo de tiempo, entraría a la rama del no, haciendo durante los siguientes 800 ms restantes una operación tras otra. Ya que la CPU tiene una capacidad de 100 operaciones por segundo, en 800 ms podrá hacer 80 operaciones.

c) Retardo fuera de la condición

En este caso, el retardo se realiza después de cada cálculo de movimiento, es decir, se cumpla o no la condición. Debido a esto, se haría 1 operación cada 200 ms, con la única diferencia de que en la primera, además, se enviaría el evento de teclado. Esto da un total de 5 operaciones y un 5% de CPU usado. Esto se debe a que si se observa la figura 3.12 b), se ve que independientemente de que se cumpla o no la condición, antes de realizar la siguiente operación se produce un retardo.

3.2.6 Amplitud de los movimientos

Al pensar en discapacidades motoras, es inevitable pensar en los múltiples tipos que hay y en los diferentes grados detrás de cada uno. Esto implica que cada persona puede tener necesidades muy distintas en lo referente a qué movimientos puede realizar y con qué amplitud. Pensando en esto, se ha implementado el parámetro que se ve en la figura 3.13, que permite al usuario seleccionar dicha amplitud, en otras palabras, cómo de grande ha de ser el movimiento para que este sea considerado válido. Este parámetro es denominado desplazamiento.

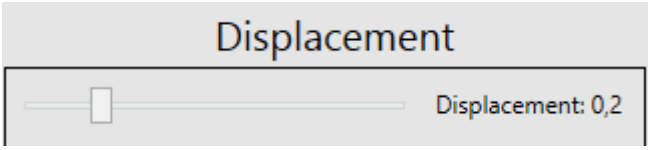

Fig. 3.13 Representación del slider que permite seleccionar el parámetro desplazamiento

En el apartado 3.2.2. "Uso de las coordenadas para la detección del movimiento", se explicó que se calcula constantemente la distancia en uno de los ejes entre dos huesos de interés para determinar la detección de movimiento. Este parámetro desplazamiento, es el que determina esa distancia. Se puede seleccionar entre 0 y 1, aunque configurarlo por debajo de 0,1 es desaconsejable, ya que la distancia mínima entre los huesos de interés ronda ese valor. Del mismo modo, valores más allá de 0,5 no son factibles para su realización. Técnicamente estos valores se expresan en metros, pero de una forma bastante relativa, ya que se calculan en base a la estimación de la distancia que hace la Kinect. En otras palabras, en base a la distancia que tiene el usuario con la Kinect, determina cual es la distancia de estos desplazamientos en metros.

Se han programado por defecto los valores óptimos de desplazamiento para personas sin problemas motores, por lo que se recomienda que, en primer lugar, el usuario compruebe si le son válidos estos, antes de intentar configurar unos propios, en el caso de tener algún tipo discapacidad o limitación a la hora de realizar los movimientos.

3.2.7 Aplicación de hilos para la detección de los movimientos

La Interfaz está pensada para que cualquier aplicación o juego del ordenador se pueda usar con movimientos corporales prescindiendo del teclado. Como no se ha realizado para una aplicación específica, es posible que alguna aplicación requiera varios movimientos al mismo tiempo, es decir, eventos de varias "teclas" simultáneamente. Para que esto sea posible, cada movimiento debe ser tratado en un proceso paralelo e independiente. Si todos los elementos fuesen tratados en un único proceso, debido a que solo se puede ejecutar una línea de código en cada instante de tiempo, por lo que mientras se estuviese ejecutando el código de uno de los movimientos, sería imposible atender a los demás. La solución tomada ha sido la de programar la mayor parte del código en un proceso padre y cada uno de los movimientos en un hilo independiente como un proceso hijo.

Implementar cada movimiento en un hilo, no solo permite el uso de varios movimientos simultáneos, también mejora significativamente la fluidez del programa, así como su rendimiento y consumo de recursos, al poder gestionarse mejor qué parte del código se ejecuta en cada momento.

La implementación de los hilos en C#, se ha realizado con la función `Thread()` de la librería *System.Object* de Microsoft [22]. Esta función se ha explicado antes en relación con los retardos, pero, en esta ocasión, sí se va a asignar una rutina de trabajo al hilo mediante un código, cosa que no se hace al usarlo como retardo. El proceso no es complejo, pero sí tiene muchas consideraciones a tener en cuenta.

En primer lugar, como se ha visto en el capítulo de antecedentes, en C# la interfaz gráfica y el código principal se comunican mediante eventos. Aquí se encuentra un problema a

resolver: aunque el proceso padre sí es capaz de obtener la información de la interfaz gráfica mediante dichos eventos, es decir, el valor de los retardos y desplazamiento, esto no sucede con los hijos o hilos creados para cada movimiento. Para solucionar este problema, y que los hijos puedan acceder a la información de la interfaz gráfica, para obtener los datos configurados del retardo, amplitud y evento actual de teclado, hay que crear variables globales. Creando estas variables en el proceso padre y actualizándolas en el tiempo, se garantiza que todos los hilos puedan obtener la información que necesitan. En la figura 3.14, se puede observar un diagrama sobre el flujo de información entre la interfaz, el proceso padre y los hijos.

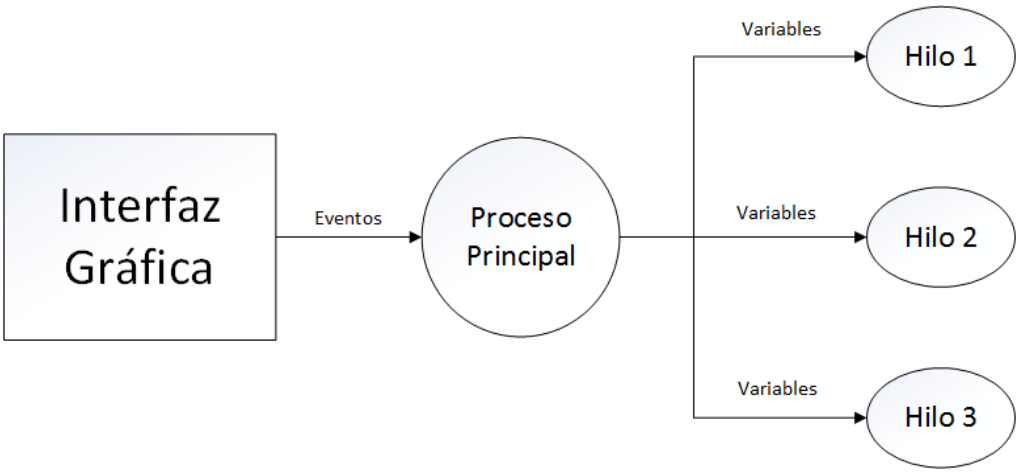

Fig. 3.14 Flujo de información entre interfaz, proceso padre y procesos hijos

Tras entender cómo se intercambia la información, se va a proceder a explicar la creación de los hilos. En primer lugar, hay que crear, como variable global, un objeto de la clase hilo con la siguiente línea de código:

```
Thread m1;
```

En este caso, se está creando un hilo con el nombre m1. Es muy importante que dicho hilo se declare como variable global; de no ser así, algunos de los métodos no podrían trabajar con ellos o con la información que generen.

Posteriormente, hay que instanciar el hilo, es decir, decirle qué rutina ha de seguir, como se observa en la siguiente línea:

```
m1 = new Thread(new ThreadStart(Movimiento1));
```

En este caso, al hilo m1 se le ha asignado la rutina Movimiento1, es decir, cada vez que se haga referencia al proceso m1, este estará ejecutando el segmento de código indicado en m1. Para que el hilo sepa cuál es la rutina que ha de realizar, esta debe de estar implementada en una función con el mismo nombre que la rutina que se le ha pasado como parámetro. En la siguiente línea se observa dicho código:

```
public void Movimiento1() {
      while(!parada){

      //Rutina del hilo
        }
 }
```

Un elemento muy importante de la rutina es el bucle `while(parada){}` . Parada es una variable *boolean*, que siempre será false, salvo que se dé la orden de detener el hilo. Es decir, este `while()`, garantiza que la rutina se repita indefinidamente. De no utilizarlo, la rutina solo se realizaría una vez y el hilo se cerraría.

Es muy importante destacar que, una vez asignada la rutina al hilo, este no está activo, hay que indicarle que comience. Para ello, hay que indicar lo siguiente en el código:

```
m1.Start();
```

Es muy importante entender que los hilos son procesos que se ejecutan paralelamente, por lo que hay que presentar mucha atención a su gestión o se producirá una sobrecarga de trabajo en la CPU. Para ello, es importante terminar los hilos cuando ya no se vayan a usar, o pausarlos si temporalmente son prescindibles. Para ello, se usan las siguientes líneas de código respectivamente:

```
m1.RequestStop();

m1.Sleep(300); // lo pausa 300 ms
```

Debido a que al generar los hilos se aumenta el número de procesos, estos solo se activan una vez la interfaz ya ha sido configurada y el usuario ha indicado que va a empezar a jugar. De esta forma, se garantiza la máxima fluidez durante el proceso de configuración en la interfaz gráfica.

3.2.8 Aplicación en primer plano

Uno de los objetivos del diseño es que, en la medida de lo posible, el programa pueda ser usado por una sola persona, es decir, que el propio usuario pueda configurarlo y usarlo sin ningún tipo de ayuda. Para ello, Mokey se ha programado de forma que, tras realizar la configuración correspondiente y pulsar jugar, la aplicación se sitúa en primer plano únicamente cuando el usuario se ha colocado ya delante de la Kinect. En la figura 3.15 se observa la parte de la interfaz en el nombre de la aplicación que se quiere poner en primer plano.

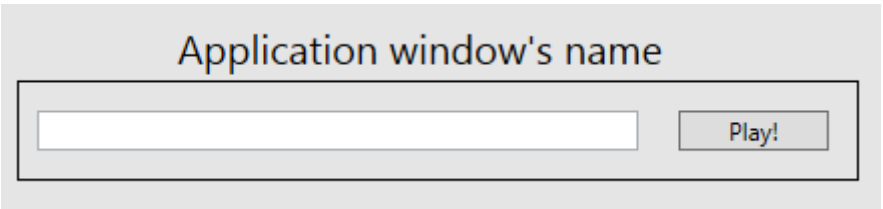

Fig. 3.15 Módulo de aplicación en primer plano

El usuario tiene que introducir el nombre de la ventana del juego que quiere tener en primer plano. Tras pulsar el botón "Play!", el mecanismo se pone en marcha. La interfaz ha sido diseñada para que, pese a darle al botón, no pase nada hasta que la Kinect no detecte al usuario. De esta manera, el usuario puede tomarse su tiempo para levantarse y colocarse. Una vez la Kinect detecta al jugador, el programa usa las funciones `FindWindows()` y `SetForegroundWindows()` de la librería DLL USER32.DLL [23]. En el siguiente código se muestra su utilización.

```
String nombre = nombre_ventana.Text;
IntPtr juego = FindWindow((string)null, nombre);
SetForegroundWindow(juego);
```

Primero se extrae el nombre que el usuario ha escrito en la ventana de la interfaz. Posteriormente, este nombre se le pasa a la función `FindWindow ()`, que va a buscarla entre las ventanas activas y va a devolver una variable de tipo *IntPtr*, con la información de dicha ventana. Tras esto, con la función `SetForegroundWindows()` y pasando como parámetro esta información, se consigue situar en primer plano.

Es importante destacar que este proceso no es necesario. Mokey funciona con la aplicación que esté en ese momento en primer plano, se haya proporcionado o no su nombre. En otras palabras, si la persona pone manualmente otra ventana en primer plano, que no se corresponde con la indicada, empezará a enviar los eventos de teclado a este nuevo programa. Por lo tanto, la función de este módulo es la de facilitar la autoconfiguración por parte de la persona que va utilizarlo. Sí es importante destacar que, aunque no se introduzca el título de la ventana y se decida poner la aplicación en primer plano manualmente, el pulsar el botón "Play!" es indispensable. Esto se debe a que tiene también, como se ha visto con anterioridad, la función de activar los hilos.

3.2.9 Función para personas en silla de ruedas

Un número muy elevado de personas con movilidad reducida, utilizan silla de ruedas, con lo cual es necesario abordar una solución para este perfil de usuario.

En la Interfaz gráfica que se puede observar en la figura 3.16, se puede observar que hay un apartado en el que, marcando una caja, se activa el modo sentado. Al ser activada, en el código se desactiva el acceso a cualquier parte del programa que haga referencia a las piernas. De esta manera, se reducen los movimientos que puede elegir el usuario de 12 a 6, pero, por otra parte, se garantiza un funcionamiento estable y sin detecciones erróneas. Los movimientos de la cabeza también son desactivados, debido a que para usarlos según están programados actualmente, se requiere un proceso de calibración que mide la altura del usuario, lo cual genera problemas al estar sentado.

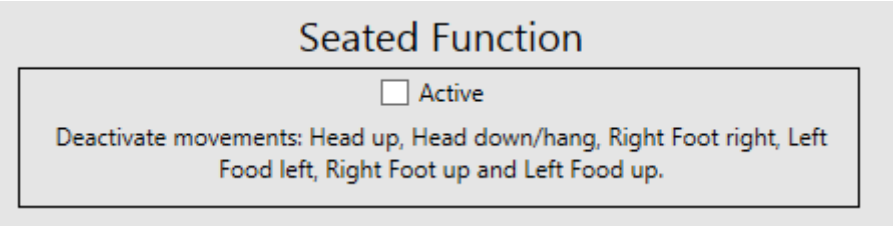

Fig. 3.16 Modo sentado de la interfaz gráfica

3.2.10 Grabación de movimientos

Con anterioridad, se ha visto que Mokey ofrece un total de 12 movimientos, de los cuales se puede configurar su amplitud. Sin embargo, esto puede ser insuficiente para personas con determinados tipos de discapacidad. Por ejemplo, un usuario puede tener hábil únicamente su brazo derecho, con los movimientos predefinidos, únicamente podría configurar tres teclas, gracias a la grabación, podría implementar hasta 4 más. También hay que tener en cuenta que los movimientos predefinidos, solo contemplan acciones sencillas como mover lateralmente, atrás o arriba. Los grabados proporcional al usuario la capacidad de hacer movimientos en cualquier otra dirección. Para intentar ofrecer la mayor versatilidad posible, la Interfaz Natural ofrece al usuario la posibilidad de que grabe cuatro movimientos propios para completar el banco ya existente. Esto puede ser de gran utilidad en personas que, por ejemplo, solo puedan mover con fluidez uno de sus brazos, ya que les permitiría asignar varios movimientos a este miembro.

El proceso de grabación se estructura en dos pasos dentro del código: en el primer paso, se produce la grabación de dicho movimiento en la interfaz gráfica del usuario; en el segundo, se activa el hilo. En la figura 3.17 se puede observar el entorno de la interfaz gráfica en el que se realiza la grabación.

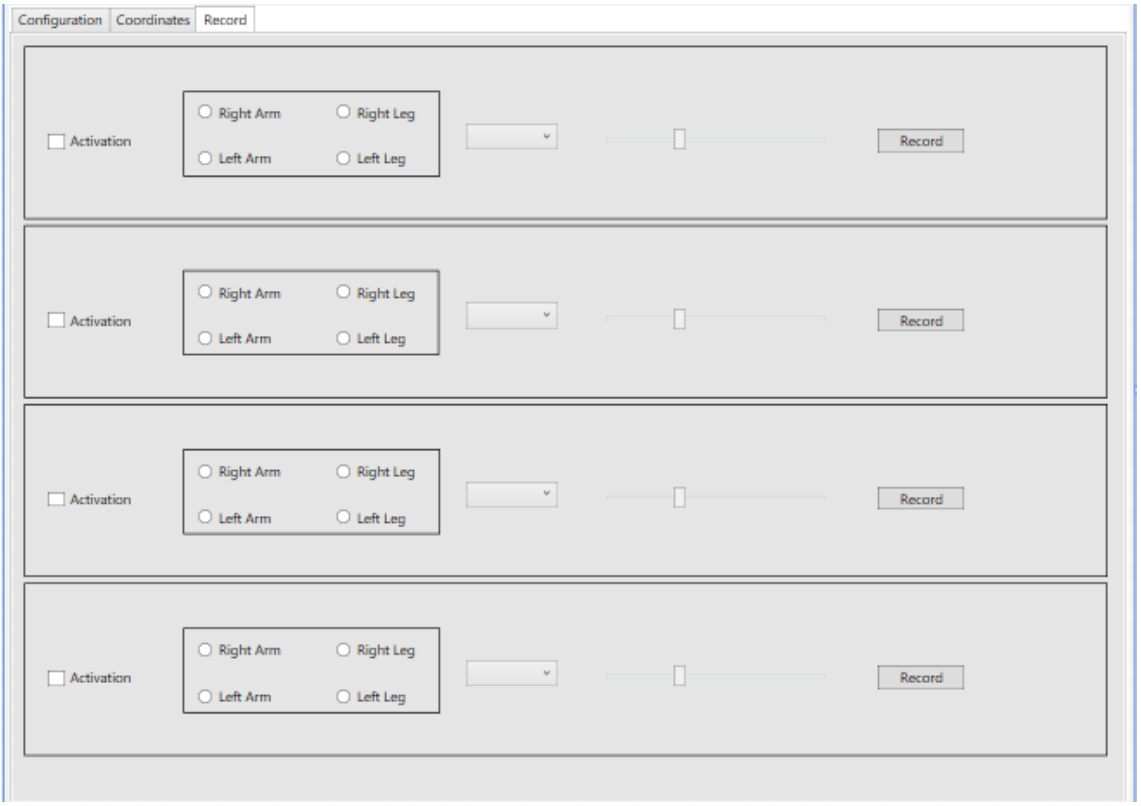

Fig. 3.17 Módulo de grabación de la interfaz gráfica

Como se puede observar en la figura 3.17, dentro del módulo de grabación hay 4 submódulos, que son los encargados de la grabación de cada uno de los cuatro movimientos posibles. En la figura 3.18 se observa la rutina de grabación.

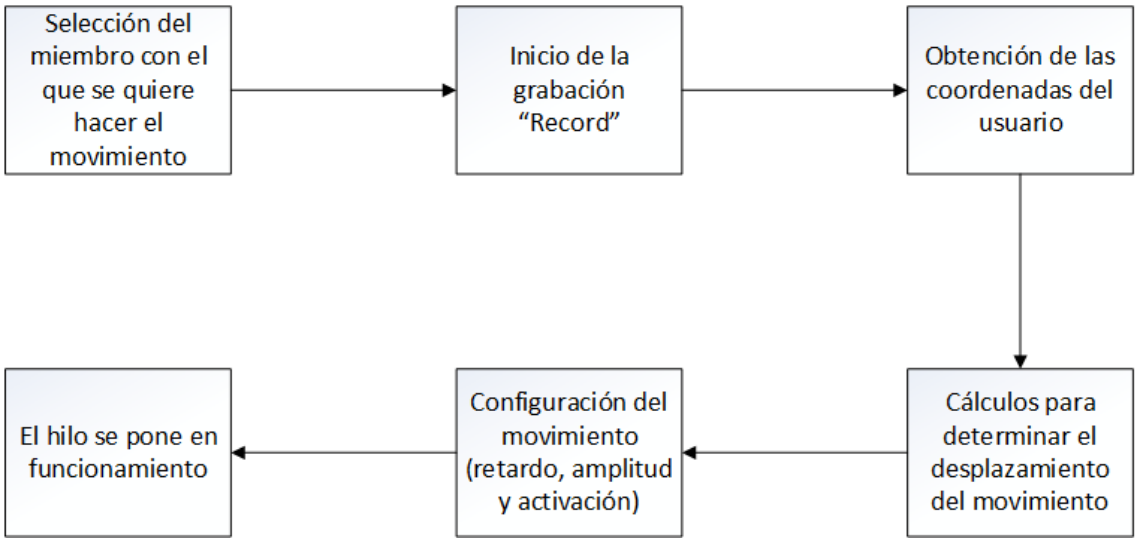

Fig. 3.18 Representación de la rutina de grabación

A continuación se va a detallar cada una de las fases vistas en la figura 3.18. En primer lugar, el usuario ha de seleccionar qué miembro del cuerpo quiere usar en el movimiento, teniendo cuatro opciones: brazo derecho, brazo izquierdo, pierna derecha y pierna izquierda. Desde el punto de vista del algoritmo necesario y cálculo computacional, esta pequeña elección por parte del usuario, simplifica ambos. Esto se debe a que, al saber en qué zona del esqueleto se va a producir el movimiento, es posible grabar solo aquella información que se sabe que es necesaria, disminuyendo de forma significativa el flujo de datos. Por otro lado, de no tener esta decisión por parte del usuario, el programa tendría que ser capaz de, con una gran cantidad de datos, determinar en dónde se ha producido el movimiento y, lo más importante, despreciar todos los movimientos residuales del usuario que con seguridad se han producido. Esto es un proceso que, llevado a un algoritmo, es extremadamente complejo, ya que se pueden dar prácticamente infinitas situaciones diferentes, que el programador nunca va a ser capaz de conseguir representar en su totalidad.

El botón de grabación "Record", activa una función que contiene la rutina de grabación. Esta capta, pasado un segundo tras pulsar el botón, la posición final del movimiento a grabar. Esta posición equivale a la distancia de desplazamiento que para los otros movimientos se configura con el deslizado. Dentro de esta función, se da la obtención de coordenadas; en función de qué miembro del cuerpo se haya seleccionado, se grabarán las coordenadas de unos huesos u otros, es decir, si se selecciona el brazo derecho, únicamente se grabaran puntos del esqueleto implicados en el brazo derecho. Siempre se van a grabar dos posiciones, la de referencia y la del hueso en su posición final del movimiento. La diferencia respecto a los movimientos predefinidos es que, en este caso, se grabará la información de los tres ejes de coordenadas (X,Y,Z) ya que no se va a limitar el movimiento a uno solo de los ejes.

Con la información de los dos huesos de interés ya obtenida, se produce un proceso de cálculos mediante el cual la función es capaz de determinar cuál es el desplazamiento del movimiento, teniendo en cuenta algunos factores que pueden llegar a crear movimientos erróneos. Para garantizar una mayor exactitud del movimiento grabado y dar al usuario la mayor versatilidad posible, el desplazamiento se mide con dos variables: valor absoluto

y signo. Debido al uso de los tres ejes, se consigue que el usuario pueda grabar movimientos en cualquier dirección posible, dando una libertad total al jugador.

Llegados a este punto, el usuario tiene que configurar exactamente los mismos parámetros que con los movimientos predefinidos: retardo, tecla y activación. En este caso, el desplazamiento desaparece porque ha sido calculado mediante la grabación. Los movimientos grabados son muy restrictivos, ya que se están definiendo con 3 valores espaciales, por lo que, si el usuario graba un movimiento en diagonal con el brazo hacia arriba, no se producirán los conflictos que se vieron en los predefinidos.

Finalmente, los hilos se activan, con el mismo funcionamiento que los predefinidos, pero con una rutina mucho más larga, debido a todas las posibilidades que hay de movimientos grabados.

Debido a la libertad que tiene el usuario en este módulo para la grabación, es posible que grabe movimientos que entren en conflicto con otro de los existentes o con otro de los grabados. Es decir, si el usuario graba, por ejemplo, dos movimientos de: brazo derecho arriba, con diferentes alturas, cuando el usuario realice el movimiento del brazo arriba más alto, estará cumpliendo los dos requisitos y se producirán los dos eventos de teclado.

3.3 Funcionamiento

En la figura 3.19 se observa un esquema del funcionamiento de Mokey. En él aparecen los algoritmos anteriormente explicados, con el fin de entender el flujo de información del programa.

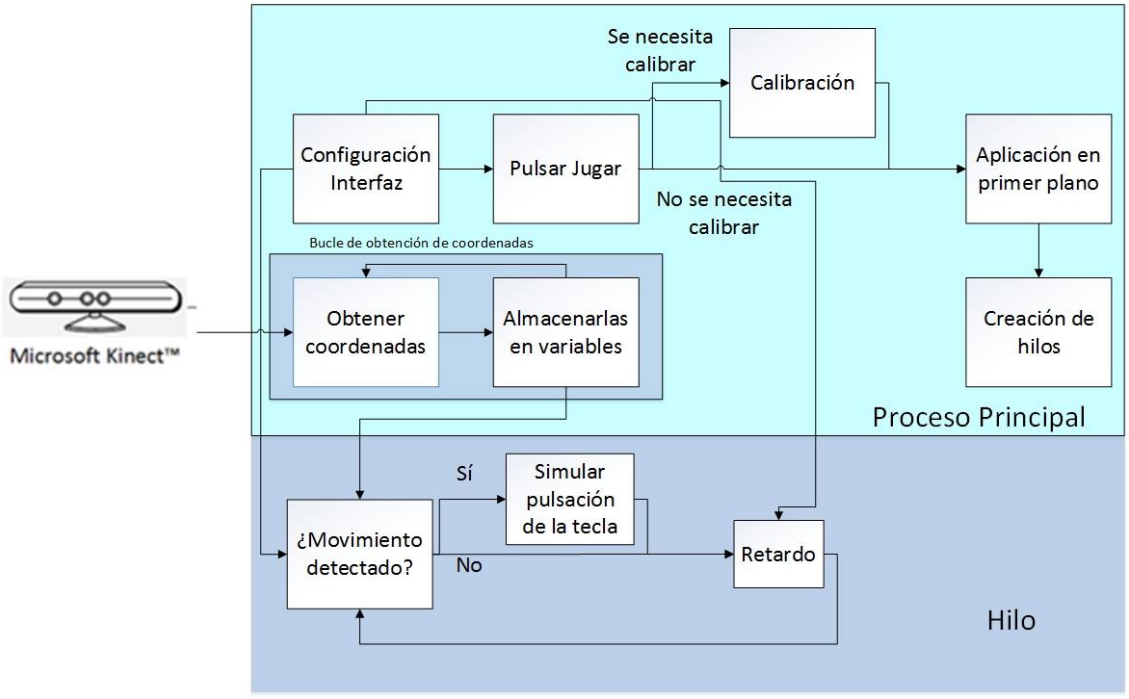

Fig. 3.19 Diagrama de funcionamiento de Mokey

En la figura 3.19 se observan dos zonas principales, una en azul claro que representa el proceso principal y otra en azul oscuro que representa uno de los hilos. Cabe destacar que en el caso de configurar más de un movimiento, habrá tantos bloques de hilos independientes como movimientos.

Dentro del proceso padre se pueden diferenciar dos etapas. La primera es la que se encarga de iniciar el programa. Tras haber realizado la configuración deseada y pulsar la tecla jugar, el programa determina si alguno de los movimientos seleccionados por el usuario requiere calibración, de ser así, esta se llevará a cabo. Posteriormente, la aplicación sobre la que se quiere usar Mokey se sitúa en primer plano. Automáticamente tras esto, los hilos de cada movimiento son creados y se ponen en funcionamiento. Toda esta rutina solo se ejecuta una vez.

La segunda etapa del proceso padre es la que se encarga de obtener las coordenadas del esqueleto, es decir, proporcionar la información del movimiento del usuario. Para ello constantemente Mokey obtiene los datos que le proporciona la SDK de Kinect y extraen las coordenadas de los puntos de interés, es decir, los huesos que están implicados en los movimientos. Tras este proceso, las coordenadas son almacenadas en variables globales. Estas variables son necesarias para que los hilos puedan ver la información que necesitan. Esta rutina se repite hasta que el usuario finalice el programa.

En el proceso del hilo se observa una única etapa. En primer lugar, se determina si el movimiento está siendo realizado. Utiliza las coordenadas almacenadas en variables para realizar las operaciones que calculan la distancia entre los huesos de interés para ese movimiento. Posteriormente compara ese resultado con el fijado por el parámetro desplazamiento por el usuario en la interfaz gráfica. En caso de que se cumpla la condición de movimiento, Mokey envía el evento de teclado. Posteriormente, se haya o no producido el evento, se producirá el retardo indicado en la interfaz gráfica, tras el cual vuelve a comprobar si se cumple el movimiento. Es importante destacar que los parámetros de desplazamiento y retardo son modificables en tiempo real**.**

Capítulo 4 PRUEBAS TÉCNICAS

A lo largo de este capítulo, se someterá a Mokey a una batería de pruebas que cuantifiquen su eficiencia y potencial de uso con diferentes aplicaciones y en diferentes situaciones. Los test realizados se van a centrar únicamente en los aspectos de cómo Mokey interactúa con el ordenador, sus recursos y con otro software. Las pruebas realizadas con usuarios referentes a la manejabilidad y compatibilidad con diferentes tipos de discapacidad motora son tratadas en [1].

4.1 Recursos del ordenador utilizados por Mokey

Un middleware es un programa que va a estar funcionando en segundo plano, para otro software que va a ser el principal. Esto, hace que haya que prestar especial atención a los recursos que el middleware consume ya que, en caso de ser muy altos, afectaría al rendimiento y correcta ejecución de la aplicación principal. Para que este análisis sea lo más detallado posible, se ha diseñado la siguiente batería de pruebas, que contemplan el consumo en las situaciones más representativas que se pueden dar en Mokey.

4.1.1 Consumo de memoria RAM

La memoria de acceso aleatorio, más conocida como memoria RAM (*Random Access Memory*), es la encargada de almacenar temporalmente todas las instrucciones que va a tener que ejecutar el ordenador, así como todos aquellos datos necesarios para la ejecución de los diferentes programas. Si la memoria RAM está llena, la velocidad de trabajo se va a ver afectada negativamente, ya que se va a tener que "hacer hueco" a las nuevas instrucciones. Si hay información en la RAM, quiere decir que se está usando, por lo que hay que decidir qué instrucciones y datos son eliminados de la memoria, pero con la premisa de que posteriormente van a tener que volver a ser reescritos. Esto tiene una clara consecuencia, cuanto más llena esté la RAM, más se verá afectado el rendimiento de un ordenador, al aumentar drásticamente los ciclos de borrado y escritura. Por todo ello, es esencial que Mokey use la menor RAM posible ya que, la aplicación en primer plano, puede llegar a necesitar un porcentaje muy alto de la memoria y el principal objetivo es garantizar su funcionamiento óptimo.

Consumo de RAM de Mokey desactivado y activado sin usuario

En La tabla 4.1 se observa el consumo de memoria RAM por Mokey en dos casos: en primer lugar, con la interfaz gráfica abierta pero sin activar el programa, en el segundo, con la interfaz abierta y el programa activado. En ambos casos no habrá ningún movimiento configurado ni el usuario se colocará frente a la Kinect. Se considera que el programa está activado cuando se pulsa el botón "Play!".

Tabla 4.1 Consumo de RAM de Mokey sin movimientos

	Desactivado	Activado (Sin usuario)
RAM (kB)	60396	61764

Los resultados que se observan en la tabla 4.1 indican que, en ambas situaciones, el consumo es similar, por lo que la activación del programa no implica un aumento de los

recursos de memoria necesarios. Esto es un dato importante ya que, al producirse esta activación, es cuando se activan todos los hilos, que son los elementos que tienen toda la carga del proceso. Del consumo de memoria, se puede decir que es bajo, de unos 60 MB. Los ordenadores de hoy en día vienen, en la mayoría de casos, con una memoria de 8 GB, por lo que se deduce que el consumo de Mokey no superaría el 1% de la RAM del equipo. Este porcentaje denota que no va a afectar de manera negativa el uso del middleware ante cualquier tipo de juego o programa, por muchos recursos que este necesite.

Para clarificar lo bajo que es este consumo, a continuación se realiza una comparativa con otros programas de uso cotidiano.

- **Google Chrome**: Una sesión abierta con una única pestaña y todas las extensiones desactivadas, ha generado un total de 5 procesos con un consumo total de 114.5 MB.
- **Skype**: Una sesión abierta, sin ninguna conversación ni llamada activa genera un consumo total de 117,2 MB.
- **Microsoft Word**: Un documento abierto de Word genera un consumo total de 53,3 MB.

De esta comparativa se puede deducir que Mokey en estado de espera consume tan poco como algunos de los programas más populares, analizados en condiciones de mínimo consumo posible.

Caso 1: Consumo de RAM de Mokey con un movimiento sin usuario, con usuario realizando movimientos y con usuario sin realizar movimientos

En la tabla 4.2 se observan los resultados del consumo de RAM en función del caso analizado. Se analizan tres situaciones posibles: en la primera, el movimiento está configurado y el software activado, pero el usuario no se pone frente a la Kinect; en la segunda, el usuario entra en el rango de la Kinect sin realizar movimientos; finalmente, el usuario realiza los movimientos asignados. La configuración usada ha sido:

- Brazo derecho arriba corresponde a la tecla A

Tabla 4.2 Consumo de RAM de Mokey en función de la configuración

	Consumo de memoria RAM (kB)		
	Sin usuario	**Con usuario (con movimientos)**	**Con usuario (sin movimientos)**
Caso 1	67036	70054	69967
Caso 2	67572	72145	73136
Caso 3	71236	72564	72569

La primera deducción que se obtiene de la tabla 4.2, es que la memoria no aumenta significativamente al configurar un movimiento. Se observa, en función de cada caso, un aumento de 7 a 10 MB; teniendo en consideración un sistema de 8 GB de RAM, este aumento es ínfimo. La procedencia de dicho incremento, está ligada, sin lugar a dudas, a la implementación de los hilos, ya que es lo único que varía respecto al caso anterior, donde no había movimientos activados y por lo tanto no había hilos creados.

Se observa una pequeña variación de 3 MB cuando el usuario entra en el rango de la Kinect. Es un valor demasiado pequeño como para determinar su procedencia con exactitud. Sin embargo, sí que hay una diferencia entre las tres situaciones: en los dos casos en los que la Kinect detecta al usuario, el flujo de datos es mayor.

Se puede deducir que no hay ninguna variación significativa en el consumo de RAM en función de que el usuario esté en movimiento o estático. Esto tiene una consecuencia directa y es que, el consumo de RAM por parte del procesado de la Kinect, no varía en función del movimiento.

Caso 2: Consumo de RAM de Mokey con 4 movimientos sin usuario, con usuario realizando movimientos y con usuario sin realizar movimientos

En la tabla 4.2 caso 2 se observan los resultados del consumo de RAM en el caso de usar Mokey con cuatro movimientos configurados. La configuración usada ha sido:

- Brazo derecho arriba corresponde a la tecla A
- Brazo derecho hacia la derecha corresponde a la tecla B
- Brazo izquierdo arriba corresponde a la tecla C
- Brazo izquierdo hacia la izquierda corresponde a la tecla D

Los datos del caso 2 de la tabla 4.2 dejan claro que, con los aumentos de movimientos, el aumento de memoria no parece muy apreciable. Este aumento se cuantifica en 2/3 MB respecto al caso anterior, que teóricamente se podría deducir que al aumentar el número de movimientos, aumenta el número de variables y, por lo tanto, la memoria que requiere. Sin embargo, hay que recordar que las pruebas son hechas en un momento determinado del tiempo y se toma un valor promedio, por lo que es un factor a tener en cuenta al ser una variación tan pequeña.

Caso 3: Consumo de RAM de Mokey con 12 movimientos sin usuario, con usuario realizando movimientos y con usuario sin realizar movimientos

En el caso 2 de la tabla 4.2 se observan los resultados del consumo de RAM en el caso de usar Mokey con cuatro movimientos configurados. La configuración usada ha sido:

- Brazo derecho arriba corresponde a la tecla A
- Brazo derecho hacia la derecha corresponde a la tecla B
- Brazo izquierdo arriba corresponde a la tecla C
- Brazo izquierdo a la izquierda corresponde a la tecla D
- Pierna derecha hacia adelante corresponde a la tecla E
- Pierna derecha hacia la derecha corresponde a la tecla F
- Pierna izquierda hacia adelante corresponde a la tecla G
- Pierna izquierda hacia la derecha corresponde a la tecla H
- Brazo derecho atrás corresponde a la tecla I
- Brazo izquierdo atrás corresponde a la tecla J
- Agacharse corresponde a la tecla K
- Ponerse de puntillas corresponde a la tecla L

Tras analizar los datos, se puede llegar a la conclusión de que, efectivamente, no afecta al consumo de RAM el número de movimientos que el usuario configure. Las variaciones

producidas, de pocos MB, se deben únicamente a las circunstancias de medición en cada caso.

Sin embargo, en esta prueba sí que hay un dato significativo. El consumo sin usuario ha aumentado unos 4 MB, cuando con anterioridad no había variado en esta situación. Este aumento, puede tener su origen en que se están empleando los dos movimientos que requieren calibración por lo que, las variables para determinarla, aumentan ligeramente el consumo. En cualquier caso, se observa que luego no es apreciable dicho aumento cuando el proceso de calibración ya no se aplica.

Conclusiones sobre el consumo de memoria RAM por Mokey

De las anteriores pruebas se puede deducir que el consumo de RAM de Mokey es suficientemente bajo como para no interferir en ningún caso con las necesidades de memoria que tenga la aplicación principal sobre la que se va a usar. También se puede afirmar con rotundidad que el número de movimientos configurados no afecta de forma apreciable al uso de la memoria.

Este apartado es de gran utilidad de cara al uso de videojuegos como aplicación principal. Por lo general, su consumo de RAM suele ser mucho más elevado que el de cualquier programa habitual, pudiendo llegar a los varios GB con facilidad.

4.1.2 Consumo de CPU

La unidad central de procesamiento, más conocida como CPU (*Central Processing Unit*), es la encargada de realizar todos los cálculos o instrucciones de un ordenador u otros dispositivos electrónicos. Cuanto más potente es un procesador, más operaciones por segundo puede realizar y, por extensión, más aplicaciones se podrán ejecutar y más potentes. De nuevo, es esencial que Mokey utilice el porcentaje de CPU más bajo posible, ya que se va a ejecutar en segundo plano junto a otra aplicación. En el capítulo 3 se ha visto qué métodos se han utilizado para limitar el número de cálculos que son necesarios. En este apartado, se va a comprobar si dichas medidas son efectivas.

Para poder determinar correctamente cuál es el consumo real de Mokey y si se ve afectado por la limitación de operaciones impuesta en el programa, como se ha visto en el capítulo 3, se van a realizar las pruebas sobre dos CPU diferentes que se muestran a continuación:

CPU 1

- Intel (R) Core 2 6400 2 núcleos a 2,13 GHz

CPU2

- Intel (R) Core i7-4790 4 núcleos y 8 subprocesos por núcleo a 3,6 GHz

La CPU 1 se corresponde con un ordenador de hace ya varios años, siendo este representativo de las condiciones menos ventajosas que pueden darse en los ordenadores domésticos. La CPU 2, representa un ordenador de último momento y, por lo tanto, un reflejo de lo que se puede encontrar en las casas de los usuarios con ordenadores desde hace dos años en adelante.

Consumo de CPU de Mokey desactivado y activado sin usuario

En la tabla 4.3 se observa el consumo de CPU de Mokey sin ningún movimiento configurado. La configuración es la misma que la observada en la tabla 4.1.

Tabla 4.3 Consumo de CPU de Mokey sin movimientos

	Desactivado	Activado (Sin usuario)
CPU 1 (%)	14	14
CPU 2 (%)	3.3	3.4

En el caso de la CPU 1, la menos potente, se observa un consumo de CPU del 14% de su capacidad de procesado. No se observan variaciones al activar la aplicación, por lo que se deduce que no se ve afectado por la activación de los hilos mientras no hay gente delante de la Kinect.

Si se tiene en cuenta que el consumo de CPU por parte de las aplicaciones más comunes mientras están en espera, es tendiente al 0%, se puede decir que el consumo de Mokey es significativo. Esto se debe a que es un programa que se basa en la repetición de operaciones dentro de bucles, por lo que el gasto computacional es alto.

En el caso de la CPU 2, que se corresponde con un ordenador de última generación, el consumo se reduce a prácticamente la cuarta parte, teniendo un valor del 3,3%, que se puede afirmar de forma categórica que no representa ningún problema para el uso de cualquier otra aplicación por potente que sea.

Caso 1: Consumo de CPU de Mokey con un movimiento sin usuario, con usuario realizando movimientos y con usuario sin realizar movimientos

En la tabla 4.4 se observa el consumo de CPU de Mokey en función de su configuración. La configuración para cada caso, es la misma que en los casos de la tabla 4.2.

Tabla 4.4 Consumo de CPU de Mokey en función de la configuración

		Sin usuario	Con usuario (con movimientos)	Con usuario (sin movimientos)
Caso 1	**CPU 1 (%)**	15	39	41
	CPU 2 (%)	3.4	9.9	10.4
Caso 2	**CPU 1 (%)**	14	40	41
	CPU 2 (%)	3.3	9.8	10.2
Caso 3	**CPU 1 (%)**	14	39	42
	CPU 2 (%)	3.2	9.3	10.6

Hay varias conclusiones importantes que se pueden obtener del caso 1 de la tabla 4.4. En primer lugar se observa que, con los movimientos ya activados, en ninguna de las dos CPU hay una variación significativa respecto al caso anterior mientras el usuario no está delante de la Kinect.

En cuanto la Kinect detecta al usuario, se observa un aumento bastante importante. En el caso de la CPU 1 sube hasta el 39%, en otras palabras, duplica ampliamente la capacidad de procesado. Es fácil deducir que esto se debe a la necesidad de procesado de las

funciones de la SDK de Kinect para obtener las coordenadas ya que, al salir de la zona de detección de la cámara, vuelve a bajar el porcentaje de uso.

Cuando el usuario no está delante, las operaciones exclusivas de Mokey sí que se están realizando, por lo que este 20% de CPU es exclusivamente debido a la SDK de la Kinect, y sería requerido por cualquier aplicación que utilice su detección de esqueleto.

En el caso de la CPU 2, la representativa del mercado actual de los ordenadores, se observa de nuevo que el consumo es la cuarta parte, en torno al 10%, lo que, aunque no se puede decir que no sea significativo, sí se puede afirmar que es suficientemente bajo como para que ninguna aplicación tuviese problemas en funcionar junto al middleware.

Cabe destacar un suceso aislado: en el instante en el que el usuario se pone frente a la Kinect, se produce un pico aislado, de uno o dos segundos, en el consumo de CPU, llegando en el caso de la CPU 1 hasta al 70%. Sin embargo, como sólo se produce en un momento aislado, sobre la aplicación principal el efecto puede considerarse despreciable.

Caso 2: Consumo de CPU de Mokey con 4 movimientos sin usuario, con usuario realizando movimientos y con usuario sin realizar movimientos

En el caso 2 de la tabla 4.4 se observa el consumo de CPU de Mokey con cuatro movimientos configurados.

Los datos indican que no hay un aumento del consumo de CPU al aumentar el número de movimientos configurados. Esto indica que las operaciones realizadas por los hilos, para determinar en tiempo real si se cumple una condición, no tienen un gasto computacional significativo. Esto se debe a la gestión de los retardos en dichos hilos, como se vio en el capítulo 3. Sin esta gestión, el porcentaje de CPU sería cercano al 100% de la CPU.

En la CPU 2, se observa de nuevo un consumo similar al anterior caso del 10%, dejando capacidad suficiente de procesado para cualquier aplicación actual.

Caso 3: Consumo de CPU de Mokey con 12 movimientos sin usuario, con usuario realizando movimientos y con usuario sin realizar movimientos

En el caso 3 de la tabla 4.4 se observa el consumo de CPU de Mokey con doce movimientos configurados.

Los resultados ratifican las conclusiones de la anterior prueba, el número de movimientos configurados no está directamente vinculado al consumo de CPU.

Conclusiones sobre el consumo de CPU por Mokey

En el caso del consumo de CPU, lo primero que se ha de decir es que sí es suficientemente significativo como para ser tenido en cuenta. El gasto mientras Mokey está en estado de espera es superior al de las aplicaciones más usuales, que suele tender a 0%, aunque no a niveles preocupantes.

Se puede determinar que, cuando el usuario está dentro del rango de la Kinect, es cuando se produce una mayor carga computacional, siendo esta debida a la generación del esqueleto por parte de la SDK de Kinect. Se producen picos de gasto de CPU en los instantes en los que el usuario se sitúa frente a la cámara, llegando en el caso de la CPU 1 al 70% en periodos inferiores a 2 s.

Pese a que, como se ha visto anteriormente, el gasto es entorno al 40% en la CPU 1, en los ordenadores actuales se puede afirmar categóricamente que el gasto computacional es totalmente compatible con cualquier otra aplicación que se quiera usar en primer plano.

4.2 Interacción de Mokey con otras aplicaciones

Mokey es un Middleware, por lo tanto, es importante analizar su rendimiento junto a otras aplicaciones. Los análisis de este apartado se van a basar únicamente en lo referente a uso de recursos del ordenador, así como limitaciones debidas a los algoritmos implementados. Las pruebas de uso de aplicaciones con usuarios mediante Mokey son analizadas en [1].

En la actualidad, la mayoría de los ordenadores son portátiles, por lo tanto, se van a realizar estas pruebas en uno. De esta forma, los test se asemejarán más con los medios reales de los que dispondrán los usuarios. El portátil usado dispone de 8 GB de memoria RAM y un procesador Intel (R) Core i7-5500U con dos núcleos con 4 subprocesos por núcleo a 2,4 GHz. En la tabla 4.5 se muestran unas mediciones de Mokey en este ordenador con un movimiento configurado para tenerlo como referencia.

Tabla 4.5 Consumo de RAM y CPU de Mokey en el portátil utilizado

	Sin usuario	Con usuario (Estático)	Con usuario (Dinámico)
RAM (kB)	67823	70675	71189
CPU (%)	4,9	14,9	15,1

Las pruebas que se muestran a continuación, han sido realizadas con dos personas con el siguiente protocolo:

1. Análisis de qué funciones del juego se pueden realizar con Mokey.
2. Configuración de la totalidad de dichas funciones en Mokey (o las máximas posibles).
3. La segunda persona completará las funciones que no sean posibles mediante Mokey, principalmente el uso de ordenador.

El uso de recursos de los programas varía mucho a lo largo del tiempo y en función de su uso. Para intentar buscar la mayor exactitud, en cada caso se probarán configuraciones comunes de los programas a utilizar y se esperará a que el consumo de CPU y RAM estén estabilizados.

4.2.1 Pruebas de rendimiento junto a otras aplicaciones

Google Chrome

Google Chrome es el navegador web más utilizado en el mundo. Permite el acceso a una cantidad ingente de información, por eso ha sido seleccionado como una de las aplicaciones a probar. Para analizar una situación real, se abrirá el navegador con 4 pestañas y dos extensiones (programas de Google Chrome) activas. En la tabla 4.6 se puede observar el consumo.

Tabla 4.6 Consumo de RAM y CPU usando Chrome con Mokey

	Chrome	Chrome + Mokey
RAM (MB)	355,9	427,1
CPU (%)	0,2	15,3

Observaciones: No se ha producido ninguna incidencia durante la prueba. Se puede manejar Google Chrome con Mokey pero con limitaciones. El navegador Google Chrome permite su manejo mediante teclas, aunque esto es mucho menos intuitivo que usar el ratón. El consumo de memoria RAM de Chrome varía mucho en función de las pestañas abiertas, así como de los *plugins* instalados. Debido a esto, sus requerimientos de RAM tienen grandes fluctuaciones.

De los datos de la tabla, se puede deducir que el consumo de RAM y CPU son perfectamente compatibles con el uso de Mokey. El problema de recursos de Chrome se encuentra en su uso excesivo de memoria RAM, lo cual no es un problema para Mokey, ya que como se ha visto en el anterior apartado, su consumo de RAM es de unos 70 MB, lo que no dificulta el uso del navegador. Gracias a los atajos de teclados se puede obtener información sencilla de múltiples webs, sin embargo, no es posible interactuar con webs complejas debido a la falta de un movimiento que sea capaz de simular el ratón en Mokey.

Google Chrome se puede considerar una muestra del resto de navegadores webs, por lo que se puede decir que Mokey sirve para navegar por webs sencillas que el usuario tenga como marcadores. Por lo tanto cualquier usuario podría utilizar sin problemas aquellas páginas que consulte usualmente, es decir, navegar entre diferentes pestañas.

Avast Antivirus

Avast es uno de los Antivirus gratuitos más utilizados hoy en día. Permite garantizar la seguridad e integridad de todos los datos del ordenador, así como el correcto funcionamiento del sistema operativo ante posibles ataques. Para comprobar el consumo real, los datos de la tabla 4.7 se corresponden con Avast realizando un análisis al equipo.

Tabla 4.7 Consumo de RAM y CPU usando Avast con Mokey

	Avast	Avast + Mokey
RAM (MB)	3,2	74,2
CPU (%)	0,2	15,3

Observaciones: No se ha producido ninguna incidencia durante la prueba. Es posible manejar el programa con total funcionalidad, gracias al teclado. Por lo general, se cree que un antivirus en funcionamiento ralentiza el ordenador. Esto se debe a que el buffer de lectura de disco se sobrecarga mucho a leer todos los archivos a analizar. Sin embargo, esto no interfiere con Mokey.

Como se puede comprobar en la tabla 4.7, el consumo de Avast es absolutamente residual en lo referente al consumo de RAM y CPU. Ha sido también interesante el comprobar

que, en este caso, el manejo del programa es posible únicamente con las 4 flechas del teclado y la tecla "enter". Mokey permitiría gestionar de forma sencilla la seguridad de un sistema informático por parte de una persona con movilidad reducida.

En este caso, Avast no es representativo de todos los sistemas de antivirus. Se han realizado comprobaciones con otros sistemas y no ofrecen la misma manejabilidad únicamente con las teclas.

Skype

Skype es uno de los programas de mensajería más comunes en la actualidad. Permite compartir mensajes de texto, llamadas, videollamadas y hasta conectar con teléfonos móviles o fijos. En la tabla 4.8 se muestran los resultados del consumo de Skype durante una llamada de audio.

Tabla 4.8 Consumo de RAM y CPU usando Skype con Mokey

	Skype	Skype + Mokey
RAM (MB)	110,3	181,5
CPU (%)	2,3	17,4

Observaciones: No se ha producido ninguna incidencia durante la prueba. Utilizando los "atajos de teclado" que proporciona Skype y que se pueden configurar en función de las necesidades del usuario, se puede realizar una llamada o una videollamada sin ningún tipo de problema usando únicamente Mokey. Sin embargo, no se pueden configurar suficientes teclas como para poder escribir un mensaje. El consumo de RAM puede variar significativamente en función del servicio usado y la calidad, pero en ningún caso interfiere con Mokey al ser el consumo de este como ya se sabe muy bajo.

Como se puede observar en la tabla 4.8, el consumo de Skype es perfectamente compatible con el uso de Mokey. Del mismo modo, la configuración de Mokey es suficiente para poder realizar llamadas y videollamadas. Cobra especial importancia que se puedan realizar llamadas, ya que una persona discapacitada podría realizar ella sola una llamada tanto a otro Skype como a un teléfono en caso de necesitar contactar con alguien para pedir ayuda.

Hay otros sistemas de mensajería y los resultados del Skype se pueden hacer extensibles a la práctica totalidad de ellos.

Microsoft Word

Microsoft Word es el procesador de textos más extendido en los ordenadores con Windows. Permite crear y leer documentos de texto en múltiples formatos. Los resultados de la tabla 4.9 muestran el consumo de un documento de Word de 10 hojas con texto e imágenes.

Tabla 4.9 Consumo de RAM y CPU usando Word con Mokey

	Word	Word + Mokey
RAM (MB)	41,9	113,1
CPU (%)	0,3	15,4

Observaciones: No se ha producido ninguna incidencia durante la prueba. Es posible moverse por el menú y utilizar múltiples herramientas del programa. Sin embargo, no es posible asociar a Mokey suficientes teclas como para poder escribir texto.

De los datos de la prueba se determina que, a nivel de rendimiento, Mokey y Word son compatibles a nivel de uso de recursos. Sin embargo, únicamente se puede decir que Mokey es útil para gestionar con facilidad la lectura de documentos y pequeñas acciones de edición de formato. Crear textos es inviable debido a la gran cantidad de teclas que serían necesarias para la configuración.

Microsoft Word es una muestra representativa del resto de editores de textos. Se puede hacer extensible que, con la mayoría de editores, se podría leer documentos usando Mokey, pero no crearlos.

Tetris

Tetris es uno de los juegos que más historia tienen en el mundo de las consolas y ordenadores. Ha sido elegido por representar un modelo de juego simple de fácil manejo, que necesita pocos recursos. En las pruebas con usuarios realizadas en [1], este ha sido el programa seleccionado desde las primeras pruebas, ya que es bastante intuitivo para los usuarios y, por lo tanto, no requiere de un proceso de adaptación grande. En la tabla 4.10 se muestran los datos de consumo de una partida activa.

Tabla 4.10 Consumo de RAM y CPU usando Tetris con Mokey

	Tetris	Tetris + Mokey
RAM (MB)	1,8	73
CPU (%)	0,2	15,3

Observaciones: No se ha producido ninguna incidencia durante la prueba. El juego es totalmente funcional con Mokey.

Como se ve en la tabla 4.10 el consumo de recursos del Tetris es tan residual que Mokey no le afecta negativamente. Este juego es idóneo para introducirse en el manejo de Mokey, debido a su sencillez y a que necesita pocos movimientos.

Minecraft

Minecraft representa uno de los videojuegos de ordenador más versátiles del mercado. Permite al usuario crear sus propios mundos o jugar en otros ya existentes. Debido a esa versatilidad, es difícil determinar una configuración considerada usual para el programa. Se ha tomado la decisión de utilizar un mundo ya existente con una configuración gráfica estándar. En la tabla 4.11 se muestran los datos de consumo de una partida activa.

Tabla 4.11 Consumo de RAM y CPU usando Minecraft con Mokey

	Minecraft	Minecraft + Mokey
RAM (MB)	486,7	557,9
CPU (%)	32,6	47,7

Observaciones: Se han producido ralentizaciones momentáneas del ordenador cuando el personaje se desplazaba por terrenos que debían ser generados. Es posible jugar de forma independiente realizando una configuración adecuada en el Minecraft, pero sin poder usar la rotación de cámara, para la que es indispensable el ratón. Sin embargo, esto no impide jugar con el juego libremente. También hay que destacar que no hay movimientos suficientes para todas las funciones de Minecraft, pero sí llega para las opciones más usadas. Minecraft es un juego que, pese a sus simples gráficos, se basa en la creación constante de nuevo terreno de juego. Esto genera unos picos de consumo de CPU muy grandes cuando el jugador está descubriendo nuevos terrenos, ya que estos han de ser generados en ese momento. En estos momentos se pueden producir picos de CPU de más del 70%.

Los datos de esta prueba han sido curiosos. La jugabilidad ha sido mucho mayor a la esperada en el comienzo de la misma, lo cual lo hace idóneo para poder ser usado por gente discapacitada. En cuanto al consumo, cuando el mundo está estabilizado, los porcentajes de CPU son totalmente compatibles con Mokey, sin embargo, en el momento en el que el juego tiene que crear terreno, se producen ralentizaciones. Hay que destacar que dichas incidencias se pueden llegar a dar también con facilidad al jugar a Minecraft sin Mokey, por lo que tampoco es un problema directamente relacionado con el middleware.

Debido a la versatilidad del juego y lo bien que parece complementarse con Mokey a un nivel de juego sencillo, podría ser una herramienta útil para crear mundos relacionados con algún tipo de acciones indicadas para la rehabilitación o ejercicios de movimiento.

League of Legends

League of Legends es el juego MMOG (*Masive Multiplayer Online Game*) más rentable y uno de los más conocidos. Permite elegir entre múltiples campeones para batallas 5 contra 5 jugadores. Los resultados de la tabla 4.12 han sido medidos en un juego normal.

Tabla 4.12 Consumo de RAM y CPU usando League of Legends con Mokey

	League of Legends	League of Legends + Mokey
RAM (MB)	1163,2	1234,4
CPU (%)	14,3	29,4

Observaciones: No se ha producido ninguna incidencia durante la prueba. Todos los controles de teclado se pueden simular, pero falta el ratón para el desplazamiento del personaje, lo que hace inviable su uso con Mokey. El videojuego está diseñado para adaptarse a la capacidad del ordenador, es decir, la calidad gráfica variará en función de la capacidad del mismo, por lo que no llegaría a sobrecargar un ordenador poco potente.

Antes de hacer la prueba, era sabido que no era posible jugar una partida utilizando Mokey. Esto se debe a que en League of Legends, el uso del ratón es la base del juego. Sin embargo, era muy interesante analizar el rendimiento del juego con el middleware. Esto se debe a que las exigencias de CPU y RAM son muy similares a las de múltiples juegos de gran calidad gráfica, por lo que, aunque a nivel funcional la falta de simulación de ratón en Mokey sea un escollo, con estos datos se puede decir que implementar la simulación de un ratón mejoraría la versatilidad de Mokey de forma real.

Conclusiones de los análisis realizados

Los programas seleccionados para las pruebas han sido elegidos como un muestreo representativo de los diferentes tipos de software. Las conclusiones que se han obtenido sobre la interacción de Mokey con otros programas son:

- Por lo general, se puede afirmar que el gasto de memoria RAM y CPU de Mokey es suficientemente bajo como para que no interfiera en el funcionamiento de cualquier aplicación principal, por potente que esta sea. Los momentos en los que se han observado picos máximos de consumo, son situaciones en las que sin Mokey también sucedería.
- Es idóneo para que, personas con movilidad reducida, puedan realizar llamadas por programas de mensajería. Esto puede facilitar mucho la vida de este sector de población, ya que les otorgaría la capacidad de poder contactar con otras personas en caso de tener algún problema, dotándoles de una mayor independencia y seguridad.
- El mayor problema que se encuentra para su interacción con juegos es la ausencia de simulación de un ratón. Muchos juegos permiten saltar estos escollos para poder realizar un uso de él, aunque siempre reduciendo su funcionalidad.
- Aunque sea algo más concreto, es muy destacable la buena sinergia que se ha encontrado entre Mokey y Minecraft. Ambos sistemas están orientados a la versatilidad y eso hace que el mundo de Minecraft pudiese ser utilizado para diseñar juegos destinados a la rehabilitación de gente discapacitada.

4.2.2 Limitaciones derivadas de los algoritmos

Limitaciones del simulador de eventos de teclado

Como se vio en el capítulo 3, la función encargada de simular el teclado es `Sendkeys()`. Aunque funciones similares han sido desarrolladas para otros lenguajes de programación, en C# presenta muchas limitaciones. Se puede simular prácticamente cualquier tecla, pero únicamente se puede simular un evento aislado, sin poder determinar su duración.

Para juegos antiguos, como simuladores de videoconsolas o juegos de recreativos, muchas veces es necesario mantener pulsada una tecla de forma continuada, es decir, enviar un pulso continuado en el tiempo. Con `Sendkeys()` esto no es posible, aunque se repita el envío del evento sin retardo, se está enviando algo similar a un tren de deltas.

Con las funcionalidades de la función en otros lenguajes, podría aumentar mucho el número de aplicaciones con las que Mokey sería compatible. A pesar de esto, las limitaciones por esta causa se dan sobre todo en aplicaciones antiguas, por lo que se puede considerar que los beneficios de la función son mucho mayores a los inconvenientes.

Limitaciones derivadas de los retardos

Hay aplicaciones que requieren de un nivel de acción-reacción muy alto por parte del usuario. Es decir, para poder realizar las acciones del juego, se requieren una sucesión ágil y varias teclas. Estas aplicaciones no son idóneas para ser usadas con Mokey, debido a las limitaciones a la hora de hacer los movimientos.

Los retardos entre pulsaciones, aunque son un buen elemento para controlar este aspecto, tienen sus limitaciones. El usuario no puede limitar el número de pulsaciones que va a ejecutar, por lo que, el uso de retardos muy bajos, puede conllevar a pulsaciones no deseadas. Del mismo modo, retardos muy largos pueden generar una falta de resolución que no permita interactuar con fluidez con el juego.

4.3 Pruebas varias

En este apartado se van a analizar pruebas que no están directamente relacionadas con los dos apartados anteriores.

4.3.1 Análisis de la gestión de recursos por parte de los movimientos grabados

Como se ha observado en el capítulo 3, el programa, además de permitir usar 12 movimientos predefinidos, permite grabar otros 4. La implementación de estos 4 movimientos es mucho más compleja y requiere de un mayor número de líneas de código y variables. Para ello, se va a medir el uso de memoria RAM y CPU para la siguiente configuración usando los movimientos predefinidos y los grabados:

- Brazo derecho a la derecha corresponde a la letra “A”·
- Brazo izquierdo a la izquierda corresponde a la letra “B”

Las pruebas han sido realizadas en el procesador Intel (R) Core i7-4790 4 núcleos a 3,6 GHz. En la tabla 4.13 se observan los resultados para la configuración con los movimientos predefinidos. En la tabla 4.14, la configuración utilizando la grabación de movimientos

Tabla 4.13 Consumo de RAM y CPU usando movimientos predefinidos en Mokey.

	Sin usuario	Con usuario (Estático)	Con usuario (Dinámico)
RAM (kB)	67841	71548	71563
CPU (%)	3,3	9.9	10.1

Tabla 4.14 Consumo de RAM y CPU usando movimientos grabados en Mokey

	Sin usuario	Con usuario (Estático)	Con usuario (Dinámico)
RAM (kB)	67872	70764	71345
CPU (%)	3,3	10	10.2

Los resultados dejan claro que los movimientos grabados, pese a utilizar hilos más complejos, no aumentan de forma significativa el consumo de recursos del ordenador. Aunque la codificación de los hilos es más elaborada en el caso de los grabados, hay que recordar que el retardo está limitando el número de operaciones por segundo. Del mismo

modo, aunque cuenta con varias variables auxiliares, estas son pocas y no guardan mucha información. Todo esto, permite implementar un sistema de grabación que aumenta la funcionalidad de Mokey sin comprometer su buen rendimiento.

4.3.2 Tiempo de configuración de la interfaz

Este aspecto será exhaustivamente analizado en [1], pero desde el punto de vista del rendimiento es interesante analizar los diferentes tiempos derivados de los procesos internos relacionados con la configuración. En primer lugar, se va a analizar cuánto tiempo transcurre desde que se abre el programa hasta que se pone en funcionamiento.

Para esta prueba, se utiliza un usuario experto, que conoce la interfaz gráfica de Mokey. Se le indica realizar la siguiente configuración:

- Brazo derecho a la derecha corresponde a la letra "A".
- Brazo izquierdo a la izquierda corresponde a la letra "B"

El tiempo transcurrido desde que se abrió la configuración hasta que el programa empezó a funcionar correctamente fue de 22 s. El tiempo que transcurre desde que el usuario pulsa el botón "Play!" hasta que el programa se pone a funcionar es instantáneo. Esto implica que la configuración no necesita de tiempos significativos para ponerse en marcha. Si es cierto que, si el usuario no está situado frente a la Kinect, Mokey no empieza a actuar hasta que detecta al usuario, pero esto ha sido programado intencionalmente así para favorecer la configuración y el uso por parte de la misma persona.

Otro factor importante es el tiempo requerido para la calibración en el caso de usar alguno de los dos movimientos que la requieren. Se produce un pequeño retardo de unos pocos segundos, pero no porque sean necesarios para el proceso. Se han programado ventanas emergentes para favorecer la comprensión del proceso por parte del usuario.

Por último, se va a analizar el tiempo que se tarda en realizar una grabación de un movimiento. Manteniendo el tiempo de configuración aparte, desde que se pulsa el botón "Record", hasta que aparece la ventana indicando la correcta configuración, el tiempo que transcurre es de 1 segundo. Este segundo ha sido de nuevo introducido como retardo para facilitar la indicación al usuario de cuándo tiene que realizar dicho movimiento.

De estas pruebas, se deduce que ninguno de los procesos de configuración internos de Mokey provoca un retardo apreciable en el funcionamiento del programa. Aquellos pequeños retardos que se puede encontrar el usuario, han sido introducidos intencionadamente para facilitar su configuración.

4.3.3 Funcionamiento de Mokey en aplicaciones Flash.

Cada día más gente tiene acceso a Internet y la cantidad de contenidos y funciones de la red es más amplia y compleja. Muchas de las páginas webs actuales tienen aplicaciones integradas que funcionan con Adobe Flash: juegos, reproductores, animaciones…

Debido al uso cada vez más extendido de estas aplicaciones y la peculiaridad de que se ejecutan en un subproceso secundario, hay que analizar su funcionamiento con Mokey. En primer lugar, hay que indicar que la función indicada para generar el teclado, `Sendkeys()`, solo hace efecto sobre la aplicación en primer plano. Si se tiene en cuenta

este hecho y que la aplicación Flash se ejecuta en un subproceso, podría pensarse que no debería funcionar.

Para comprobarlo, se ha usado Mokey en múltiples aplicaciones Flash dentro del navegador Google Chrome. Algunos ejemplos son: Candy Crash, Tetris Online, Mario Bross online…

El resultado es que la totalidad de estas aplicaciones han funcionado correctamente con Mokey. Esto se debe a que, aunque esas aplicaciones no se encontraban en el proceso que está en primer plano, el de Chrome, este es capaz de enviar la información que se produce en él a todos los subprocesos que dependen de él. El propio navegador basa su funcionamiento en la creación de múltiples hilos para gestionar todo el navegador, por lo que, de no haber sido esto así, no se podrían haber usado estas aplicaciones.

4.4 Comparativa con FAAST

Como se pudo ver en el análisis del Estado del Arte del capítulo 2, FAAST es una de las interfaces naturales más desarrollada y extendida. Esta aplicación puede ser descargada de forma gratuita desde la web del desarrollador [24] Por esta razón, se van a comparar algunos de los datos técnicos más representativos entre Mokey Y FAAST. Con motivo de garantizar la objetividad de las pruebas, las medidas de Mokey han sido realizadas de nuevo tras finalizar las de FAAST, para evitar que cualquier imprecisión eventual del ordenador pudiese afectar solo a una de las medidas. Debido a esto, puede haber ligeras variaciones no significativas con respecto a los valores mostrados en el apartado 4.1, ya que no dejan de ser valores promediados en un periodo de tiempo. Todas las medidas han sido realizadas en la CPU 1 descrita en el punto 4.1.

4.4.1 Comparación de uso de RAM entre Mokey y FAAST

En la figura 4.1 se muestra la representación del consumo de RAM de ambas aplicaciones en 5 casos con la misma configuración:

- **Caso 1:** Interfaz funcionando sin configuración de movimientos.
- **Caso 2:** Interfaz funcionando con 1 movimiento configurado, con el usuario sin moverse. La configuración es la misma que la utilizada en el caso 1 del apartado 4.1.1.
- **Caso 3:** Interfaz funcionando con 1 movimiento configurado, con el usuario moviéndose. La configuración es la misma que la utilizada en el caso 1 del apartado 4.1.1.
- **Caso 4:** Interfaz funcionando con 4 movimientos configurados, con el usuario sin moverse. La configuración es la misma que la utilizada en el caso 2 del apartado 4.1.1.
- **Caso 5:** Interfaz funcionando con 4 movimientos configurados, con el usuario moviéndose. La configuración es la misma que la utilizada en el caso 2 del apartado 4.1.1.

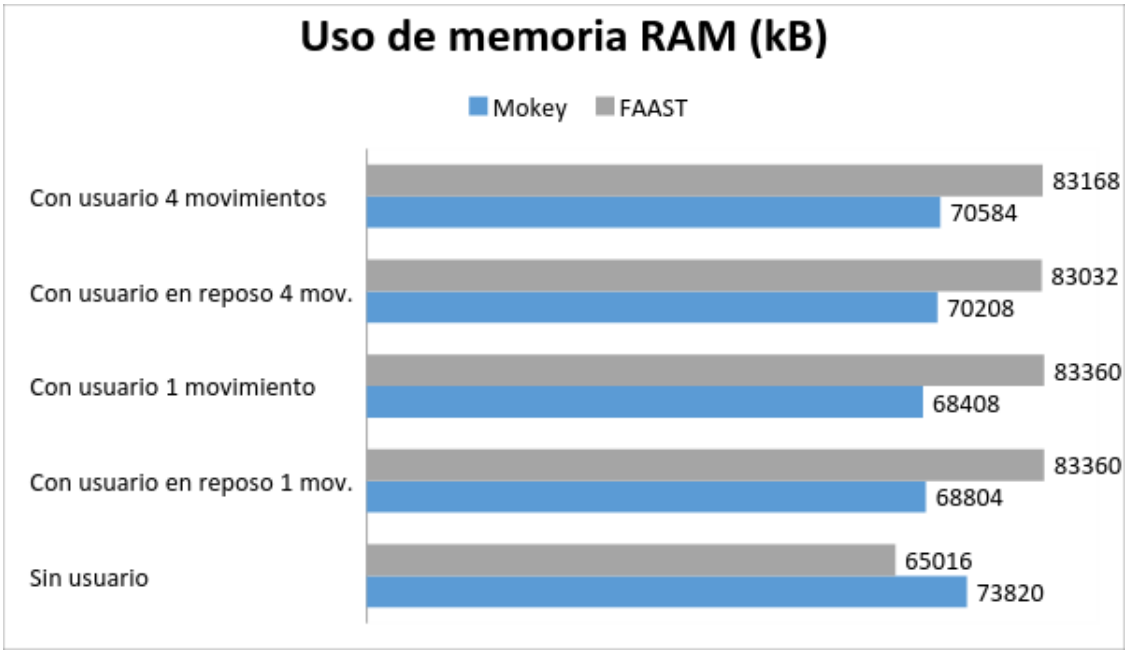

Fig. 4.1 Representación del uso de la memoria RAM tanto para el FAAST como para Mokey. Representado en kilobytes.

Como se puede observar en la figura 4.1, el gasto de memoria RAM es muy similar en ambos sistemas. La variación en función del número de movimientos también se puede considerar invariante. Teniendo en cuenta que los ordenadores actuales cuentan con 8 GB, un consumo de memoria de 70 MB o 80 MB es muy bajo, por lo que ambos sistemas son perfectamente compatibles con cualquier programa que requiera mucha RAM.

4.4.2 Comparación de uso de CPU entre Mokey y FAAST

En la figura 4.2 se muestra la representación del consumo de RAM de ambas aplicaciones en 5 casos con la misma configuración:

- **Caso 1:** Interfaz funcionando sin configuración de movimientos.
- **Caso 2:** Interfaz funcionando con 1 movimiento configurado, con el usuario sin moverse. La configuración es la misma que la utilizada en el caso 1 del apartado 4.1.1.
- **Caso 3:** Interfaz funcionando con 1 movimiento configurado, con el usuario moviéndose. La configuración es la misma que la utilizada en el caso 1 del apartado 4.1.1.
- **Caso 4:** Interfaz funcionando con 4 movimientos configurados, con el usuario sin moverse. La configuración es la misma que la utilizada en el caso 2 del apartado 4.1.1.
- **Caso 5:** Interfaz funcionando con 4 movimientos configurados, con el usuario moviéndose. La configuración es la misma que la utilizada en el caso 2 del apartado 4.1.1.

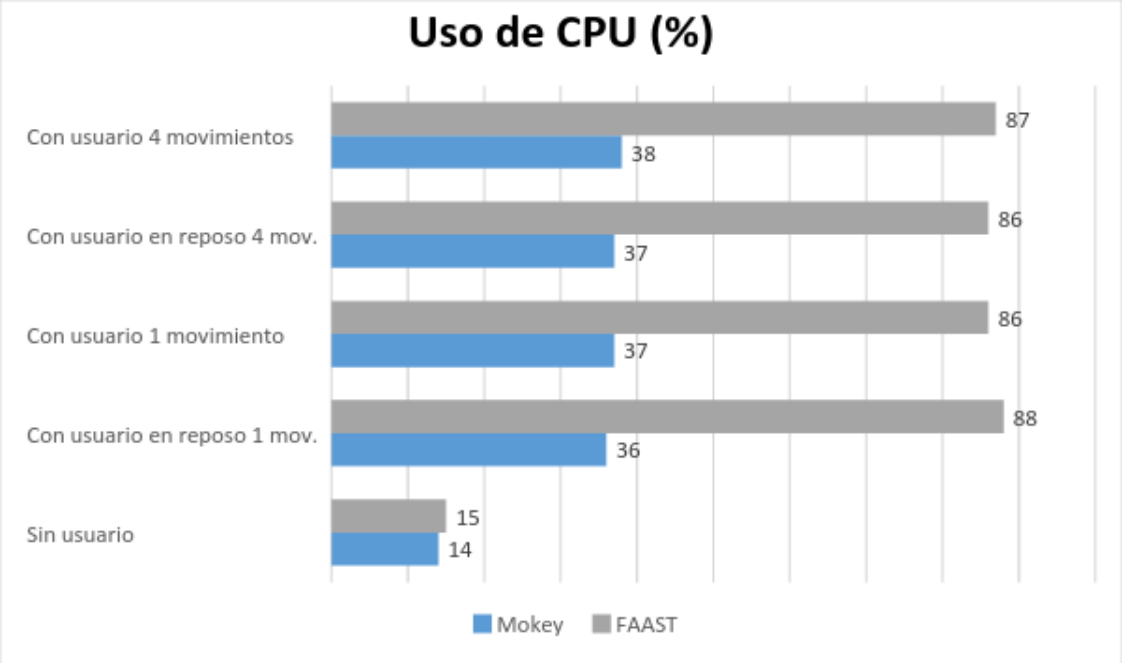

Fig. 4.2 Representación del uso de la CPU tanto para el FAAST como para Mokey. Representado en tanto por ciento.

Como se observa en la figura 4.2, el consumo de ambos sistemas en reposo es muy similar, en torno al 15%. En este caso, también se observa que la carga computacional no se ve afectada en función del número de movimientos de una manera significativa. Sin embargo, el gasto de CPU de FAAST es cercano al 90%, mientras que el de Mokey es de 37% aproximadamente. Esto tiene una consecuencia directa: FAAST no podría ser usado con ningún programa que requiriese más del 10% de la CPU, lo que limita enormemente su funcionalidad. Además, hay que destacar que, a mayores del middleware y el programa principal, el Sistema Operativo y otros programas abiertos también requieren de CPU.

4.4.3 Comparación de tiempo de configuración entre Mokey y FAAST

Las comparativas derivadas del tiempo necesario para la configuración de ambas por parte de los usuarios son tratadas en [1]. Sin embargo, hay una consideración muy importante a tener en cuenta en cómo funcionan ambos programas tras ser activados por el usuario. Se ha observado que Mokey empieza a funcionar instantáneamente, sin embargo, esto no sucede con FAAST.

Realizando las pruebas sobre la misma CPU que las anteriores y considerando solo el tiempo que transcurre desde que el usuario pulsa la tecla indicada para activar el middleware hasta que esta es funcional, “Play!” en el caso de Mokey, se ha obtenido la tabla 4.15.

Tabla 4.15 Tiempo desde inicio de la activación hasta puesta en marcha en Mokey y FAAST

	Mokey	**FAAST**
Tiempo (s)	< 1	20

Como se puede observar en la tabla 4.15, Mokey empieza a funcionar instantáneamente, sin embargo, en FAAST se observa un retardo de 20 segundos. Es difícil determinar a qué se debe esta espera, ya que en el programa no se indica nada de que se está procesando la configuración ni nada similar. También se ha observado que, a la hora de cerrar ambos programas, en muchas ocasiones FAAST se queda bloqueado impidiendo su correcto cierre, así como cualquier otra acción.

No obstante, sea cual sea la procedencia de este retardo, lo cierto es que no es tan considerable como para considerarlo un factor diferenciador crítico.

4.4.4 Conclusiones de la comparación

Desde el punto de vista de la gestión de los recursos del ordenador, Mokey es más eficiente. En ordenadores con procesadores antiguos, FAAST no podría ser usado debido a su alto consumo de procesador. En cuanto a la RAM, ninguno de los dos es un problema de compatibilidad con la aplicación principal.

Los resultados, en cuanto al número de aplicaciones que se pueden usar en ambos, son muy similares y, como se puede comprobar detalladamente en [1], los usuarios consiguen resultados muy similares con ambas herramientas.

En conclusión, ambas Interfaces Naturales ofrecen unos resultados similares de cara a las aplicaciones que pueden ser usadas con ellas. FAAST ofrece un mayor número de posibilidades en lo referente a la configuración de movimientos; por el contrario, esta configuración es muy compleja. Mokey, por su parte, ofrece una configuración más sencilla, con la posibilidad de grabar movimientos propios y gestionando de una forma más eficiente los recursos del ordenador.

Capítulo 5 CONCLUSIONES y LÍNEAS FUTURAS

En esta memoria se ha presentado el desarrollo y análisis de la Interfaz Natural Mokey.

5.1 Conclusiones

Se pueden extraer conclusiones de dos aspectos principales: en primer lugar, de las características técnicas obtenidas gracias a la implementación; en segundo, las obtenidas tras las diversas pruebas técnicas realizadas. Hablando de las características técnicas logradas de Mokey, las conclusiones son:

- Se permite el uso mediante movimientos de cualquier aplicación que requiera de un teclado para su manejo.
- Se permite al usuario elegir entre 12 movimientos predefinidos, 2 de los cuales necesitan una calibración inicial.
- Se permite al usuario determinar el retardo mínimo entre la repetición de dos eventos de teclado.
- Se permite al usuario determinar la amplitud del movimiento, es decir, cuán amplio ha de ser un movimiento para que se cumpla la condición para generar el evento de teclado.
- Se permite al usuario elegir un modo especialmente pensado para gente en silla de ruedas. Este modo gestiona internamente todas aquellas excepciones que pueden surgir de la posición del usuario. Con este modo, el usuario puede elegir entre 6 de los movimientos predefinidos.
- Se permite al usuario guardar y cargar configuraciones de la interfaz. De esta manera se consigue que, una vez el usuario tenga definidas las condiciones más idóneas para sus necesidades, no tenga que volver a buscarlas.
- Se permite al usuario seleccionar la aplicación que se va a utilizar en primer plano. De esta manera, dicha aplicación se pondrá automáticamente en primer plano cuando el usuario esté delante de la Kinect. Este aspecto favorece que sea el mismo usuario que va a usar la aplicación el que la configure.
- Se permite al usuario grabar hasta 4 movimientos a mayores de los predefinidos. Esto dota a Mokey de mayor versatilidad y, además, permite a usuarios que tienen limitaciones en el número de miembros corporales que pueden usar, configurar varios movimientos con un único miembro.

Los aspectos relacionados con su funcionalidad han sido tratados en el capítulo 4. Las conclusiones de dichas pruebas son:

- El consumo de memoria RAM de Mokey es muy bajo. Se sitúa en torno a los 70 MB, lo cual hace que sea totalmente compatible con cualquier programa principal.
- El consumo de porcentaje de CPU sí que es significativo. Sin embargo, en ordenadores actuales, se ha podido determinar que es totalmente compatible con cualquier aplicación principal. Las ralentizaciones del ordenador que se observaron en momentos puntuales, se hubiesen producido también sin usar Mokey.
- Mokey no tiene ningún impacto apreciable en otros recursos del ordenador como uso de red o buffer de escritura/lectura en el disco duro.

- Las limitaciones en el uso de aplicaciones, vienen derivadas principalmente de no simular los eventos de un ratón y del número de teclas máximas que se pueden simular, que vienen limitadas por el número de movimientos permitidos.
- Con las especificaciones actuales, permite utilizar a personas con movilidad reducida aplicaciones cotidianas, que les permiten acceder a información en internet, leer e interactuar de forma sencilla con documentos de texto y realizar llamadas mediante sistemas de mensajería instantánea.
- En cuanto a nivel de juegos en un ordenador actual, queda patente que Mokey es perfectamente compatible con cualquier videojuego por muchos recursos que este necesite. Por otra parte, en este campo, la limitación de no poder simular el ratón es el mayor obstáculo para una funcionalidad plena.
- La función que simula los eventos de teclado limita el número de aplicaciones que se pueden utilizar al no permitir generar pulsos continuos. Sin embargo, se puede afirmar que este aspecto solo afecta a aplicaciones de hace varias décadas o simuladores de dichas aplicaciones, por lo que no es un problema significativo.

Las conclusiones directamente relacionadas con los usuarios son analizadas de forma detallada en [1].

5.2 Líneas futuras

De todas las conclusiones obtenidas en el anterior apartado, se pueden observar varios puntos a tratar en líneas futuras de desarrollo. A continuación, se muestran algunas de las posibles mejoras del sistema a medio plazo:

- Permitir la simulación de eventos de ratón. Como se ha podido observar, para que la Interfaz natural sea lo más versátil posible, la simulación de un ratón es necesaria. En el lenguaje C# no hay funciones implementadas que permitan hacerlo de una forma sencilla, pero sí hay herramientas para poder diseñar funciones propias.
- Sustituir la función `Sendkeys()`. Como se ha visto, el desarrollo de esta función en C# es exiguo, por lo que presenta pequeñas limitaciones a la hora de usar determinadas aplicaciones. Con las mismas herramientas que las del punto anterior, se puede desarrollar una función propia que sea más completa y, por lo tanto, dote a Mokey de una mayor versatilidad.
- Optimizar el consumo de CPU de Mokey. Uno de los puntos más destacables en el uso de recursos de Mokey es su significativo consumo de CPU en estado de espera y durante su uso. El porcentaje referente a los cálculos para la creación del esqueleto no podrá ser modificado de ninguna manera, ya que no depende de Mokey, pero se puede buscar una optimización mayor en lo referente al consumo de comprobación de criterios que realiza el middleware.
- Optimizar los movimientos grabados. Los hilos que gestionan estos movimientos son bastante complejos y deben de considerar las posibles excepciones que ocasionaría algún error de ejecución. Optimizar este código, generaría movimientos de mayor precisión y que interfieran en la menor medida posible con otros movimientos ya creados.
- No poner limitaciones al número de movimientos grabados. Algunas aplicaciones no pueden ser utilizadas debido a que el número de teclas necesarias para su uso no puede ser configurado en Mokey. Esto podría tener solución si se da al usuario

la opción de grabar tantos movimientos como le sean necesarios. Sin embargo, esto generará seguramente varios problemas de detecciones falsas.

- No restringir los movimientos grabables a una coordenada, sino ampliar la grabación al espacio 3D.
- Añadir la posibilidad de grabar movimientos muy específicos en los que el usuario debe describir una curva o un camino concreto. Para esto hay que captar la posición en varios instantes de tiempo. Esta posibilidad llevaría consigo muchas más dificultades, como p.ej. solucionar un rango de tolerancia, visualizar los movimientos a grabar y comparar con los grabados, ajuste posterior de movimientos grabados etc.
- Gestionar las interacciones indebidas entre varios movimientos. Para esta gestión, lo más idóneo sería acotar los criterios de movimiento cumplido entre dos valores y no solamente a uno como se hace en la actualidad .Es decir, considerar cómo válidos los movimientos que estén situados entre dos puntos del espacio. Otra opción, sería realizar una comprobación de que el movimiento grabado no interfiere con alguno de los existentes e indicarle al usuario que debe de grabar otro.
- Se puede utilizar en aplicaciones en Adobe Flash. Esto podría ser muy interesante para generar un sitio web con aplicaciones especialmente pensadas para personas con movilidad reducida.

REFERENCIAS

[1] C. Lázaro, "Desarrollo del teclado virtual "MoKey" basado en gestos para personas con movilidad reducida: capa de usuario", Proyecto Fin de Grado, Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, Universidad Politécnica de Madrid, Pendiente de publicación, Previsto junio 2016.

[2] Organización Mundial de la Salud. *Informe mundial sobre la discapacidad.* Ginebra: Ediciones de la OMS, 2011.

[3] Microsoft Corporation, *Kinect Manual*, Microsoft Corporation, 2010.

[4] S. Kean. *Meet the Kinect: an introduction to programming natural user interfaces.* Technology in Action, 2012.

[5] Open Natural Interaction, "OPENNI SDK", junio 2015. [En línea]. Disponible: http://openni.ru/openni-sdk/ .

[6] D. Ramos, "Estudio cinemático del cuerpo humando mediante Kinect", Proyecto Fin de Carrera, Escuela Universitaria de Ingeniería Técnica de Telecomunicación, Universidad Politécnica de Madrid, Septiembre 2013.

[7] Microsoft Corporation, "Kinect for Windows features", junio 2015. [En línea]. Disponible: https://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx .

[8] Microsoft Corporation, "Kinect for Windows SDK 1.8", Junio 2015. [En línea]. Disponible: https://msdn.microsoft.com/en-us/library/hh855347.aspx .

[9] Organización Internacional de Normalización, "ISO/IEC 23270:2006", Septiembre 2006.

[10] Microsoft Corporation, "Visual Studio", Junio 2015. [En línea]. Disponible: https://www.visualstudio.com .

[11] E. a. Suma, B. Lange, A. S. Rizzo, D. M. Krum, y M. Bolas, "FAAST: The Flexible Action and Articulated Skeleton Toolkit," *IEEE Virtual Reality Conference*, pp. 247–248, Marzo 2011.

[12] F. Lamberti, A. Sanna, G. Paravati, C. Demartini, y P. Torino, "Endowing Existing Desktop Applications with Customizable Body Gesture-based Interfaces," *International Conference on Consumer Electronics*, pp. 558–559, 2013.

[13] S. Lee and S. Oh, "A Kinect Sensor based Windows Control Interface," *Int. J. Control Autom.*, vol. 7, no. 3, pp. 113–124, 2014.

[14] A. Kar, "Skeletal Tracking using Microsoft Kinect", 2011.

[15] L. Xia, C. Chen y J.K. Aggarwal, "Human Detection Using Depth Information by Kinect", *IEEE Computer Society Conferente*, pp. 15-22, Junio 2011.

[16] C. Sinthanayothin, N. Wongwaen and W. Bholsithi, “Skeleton tracking using Kinect Sensor & Displaying in 3D Virtual Scene”, *IJACT*, Vol. 4, No. 11, pp. 213-223, 2012.

[17] M. Martínez, F.J. Díaz, A. Tejero, F. Perozo, M. Antón y D. González, “Monitorización del cuerpo humano en 3D mediante tecnología Kinect”, *SAAEI XVIII Edition*, pp. 747-752.

[18] M. Elgendi, F. Picon y N. Magnenat, “Real-Time Speed Detection of Hand Gesture using Kinect”, *Annual Conference on Computer Animation and Social Agents, Mayo 2012*.

[19] Elg_14 M. Elgendi, F. Picon, N. Magnenat y D. Abbott, “Arm movement speed assessment via a Kinect camera: A preliminary study in healthy subjects”, *BioMedical Engineering OnLine*, Vol. 13, Julio 2014.

[20] Figura esqueleto Microsoft Developer España, “Reto SDK de Kinect: Detectar posturas con Skeletal tracking”, Junio 2015. [En línea]. Disponible: http://blogs.msdn.com/b/esmsdn/archive/2011/08/09/reto-sdk-de-kinect-detectar-poses-con-skeletal-tracking.aspx .

[21] Microsoft Corporation, “Sendkeys (Clase)”, Junio 2015. [En línea]. Disponible: https://msdn.microsoft.com/es-es/library/system.windows.forms.sendkeys(v=vs.110).aspx .

[22] Microsoft Corporation, “Thread (Clase)”, Junio 2015. [En línea]. Disponible: https://msdn.microsoft.com/es-es/library/system.threading.thread(v=vs.110).aspx .

[23] Andrea Visual Basic, “USER32.DLL”, Junio 2015. [En línea]. Disponible: http://www.andreavb.com/API_USER32.html .

[24] USC Institute for Creative Technologies, “Flexible Action and Articulated Skeleton Toolkit (FAAST)”, Junio 2015. [En línea]. Disponible: http://projects.ict.usc.edu/mxr/faast/ .

ANEXO 1 Manual de instalación y configuración

En el presente anexo se muestra el tutorial de instalación y uso de la aplicación Mokey. Este tutorial ha sido desarrollado para ordenadores con las siguientes especificaciones:

- Windows 8.1 o inferior a 64/32 bits
- Cámara Microsoft Kinect 1
- SDK Microsoft Kinect v1.8

Al utilizar la SDK oficial de Microsoft, el uso de Mokey está limitado a entornos Windows. A fecha de creación de este anexo, la última versión de dicho sistema operativo es la 8.1, por lo que no se puede garantizar que funcione en versiones superiores.

A 1. 1 Pasos previos a la instalación de Mokey

Para el uso de Mokey se requieren los siguientes elementos:

- Programa "Mokey.exe"
- SDK Microsoft Kinect v1.8
- Cámara Microsoft Kinect 1

Mokey ha sido desarrollado para la Kinect 1. En el momento de creación de este tutorial se encuentra en el mercado la Kinect 2, con la que no se puede garantizar el funcionamiento de la misma.

Antes de utilizar Mokey.exe, es necesario la instalación de la SDK Microsoft Kinect v1.8 [8]. Las versiones superiores se corresponden con la Kinect 2, por lo que no se puede garantizar su funcionamiento. Es recomendable que la Kinect no esté conectada durante este proceso de instalación.

Posteriormente, conectar la Kinect y comprobar que esta es correctamente detectada por el ordenador.

A 1 .2 Instalación de Mokey

Mokey se encuentra en un archivo ejecutable denominado "Mokey.exe". Dicho ejecutable es una versión portable, es decir, no requiere de ninguna instalación. Al hacer doble clic sobre el programa, automáticamente aparece la interfaz gráfica del mismo.

A 1. 3 Configuración de Mokey

En este apartado se va a indicar, paso a paso, cómo se ha de proceder para la realización previa de Mokey. Los pasos a seguir son los siguientes:

1. **Conectar la Cámara Microsoft Kinect 1 a un puerto USB del ordenador**. Comprobar que esta ha sido correctamente detectada. Para ello, revisar en la parte inferior derecha de la barra de tareas que no hay ningún icono de instalación de drivers y que, en el icono de "Quitar dispositivo de forma segura", aparece la cámara.

2. **Ejecutar el archivo "Mokey.exe"** haciendo doble clic sobre él. En caso de que la Kinect no esté conectada, se mostrará la ventana que se ve en la figura A.1.1.Una vez conectada, se observa la interfaz gráfica mostrada en la figura A.1.2.

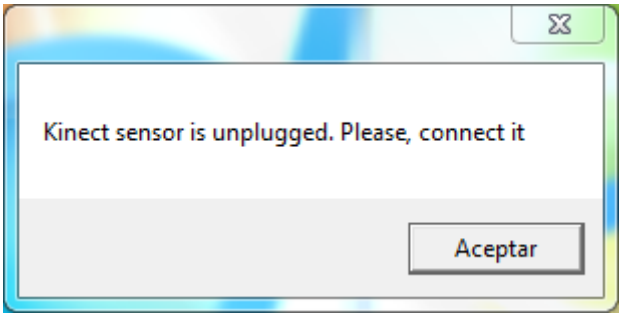

Fig. A.1.1 Ventana de información si no se detecta la Kinect conectada

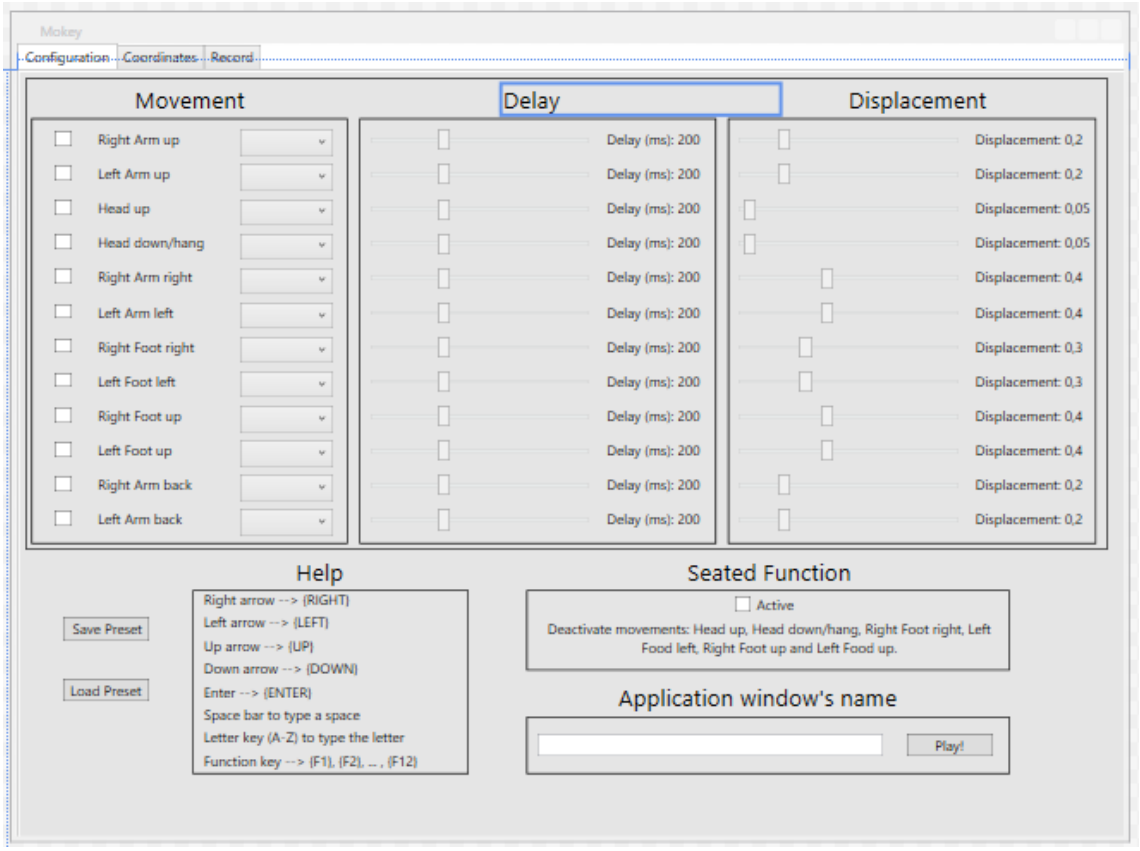

Fig. A.1.2 Interfaz gráfica de Mokey

3. **Seleccionar si se va a utilizar el modo especial para silla de ruedas**. Este módulo desactiva los 4 movimientos de las piernas y los dos que requieren calibración. Aunque el usuario los seleccione, no se ejecutarán. Para activarlo, hay que marcar la caja que se observa en la figura A.1.3

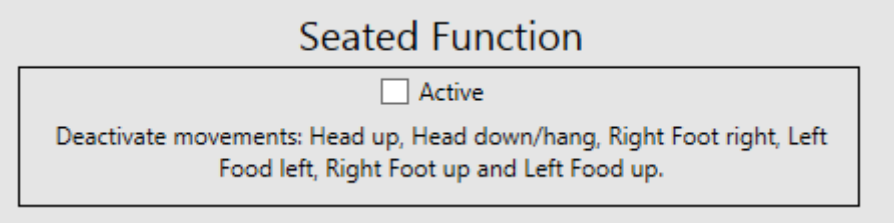

Fig. A.1.3 Módulo de selección de modo sentado

4. **Determinar qué movimientos se van a utilizar**. En la figura A.1.4 se muestra el módulo con los 12 movimientos posibles. Se deben de seleccionar, marcando la caja que se encuentra a la izquierda de los mismos, aquellos que desean ser usados.

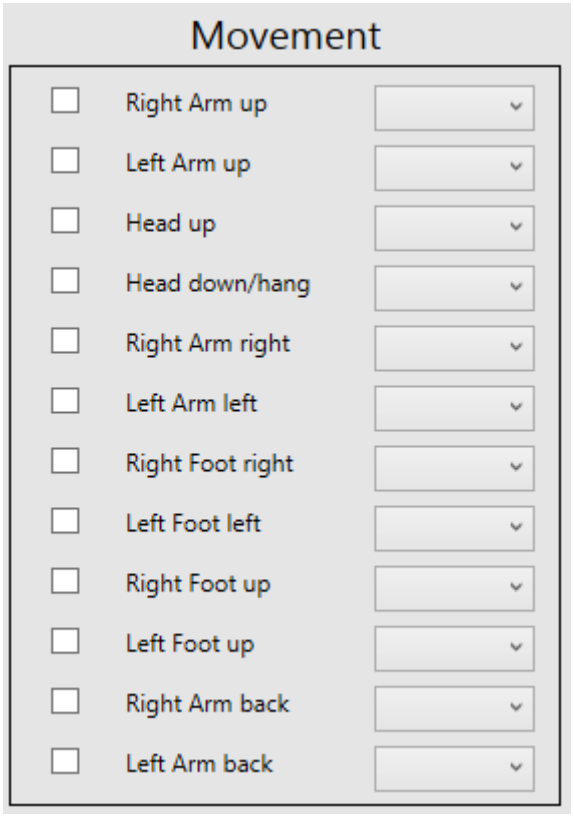

Fig. A.1.4 Módulo de selección de movimientos

5. **Asociar las teclas deseadas a cada evento**. Una vez seleccionados los movimientos a utilizar, justo a la derecha de cada uno aparece una lista desplegable para cada uno que permite seleccionar la tecla deseada, como se observa en la figura A.1.5. Se recomienda al usuario mirar la tabla incluida en la propia interfaz para ver las opciones que hay que marcar en el caso de teclas especiales como las flechas.

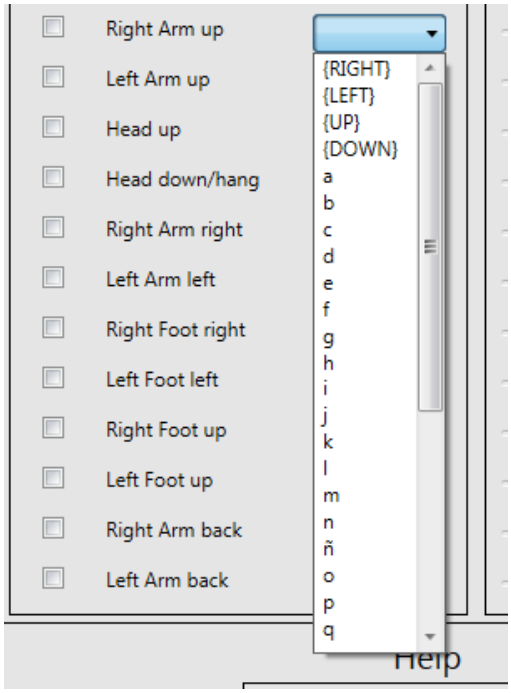

Fig. A.1.5 Módulo de selección de teclas

6. **Determinar el retardo entre pulsaciones**. Este parámetro por defecto se encuentra en 200 ms, lo que lo hace válido para la mayoría de juegos. El usuario puede variar su valor entre 0 ms y 600 ms en la barra deslizadora que se muestra en la figura A.1.6. Cuanto mayor sea, más tiempo pasará entre la repetición de dos eventos cuando un movimiento se mantiene en el tiempo.

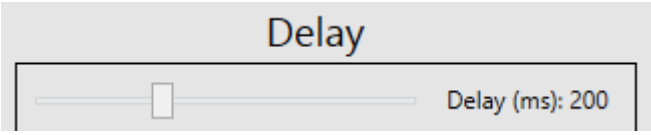

Fig. A.1.6 Módulo de selección de retardos

7. **Determinar la amplitud de movimiento o desplazamiento**. Este parámetro por defecto se encuentra en el valor más óptimo para cada movimiento. En caso de que el usuario desee variar su valor, puede hacerlo en la barra deslizadora que se muestra en la figura A.1.7. Este valor puede ser variado de 0 a 1; cuanto más próximo a 0 sea el valor, menos amplio tendrá que ser el movimiento que ha de realizar el usuario para que se simule la tecla. Por el contrario, cuando más próximo a 1, más amplio deberá ser este movimiento. No obstante, se recomienda no usar valores por debajo de 0,1 ni por encima de 0,5. Fuera de este rango, no se producen condiciones estables en los movimientos.

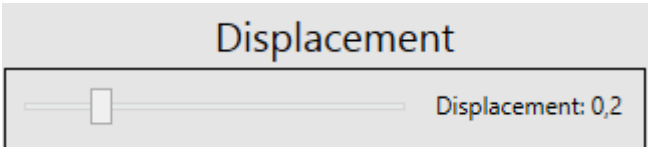

Fig. A.1.7 Módulo de selección de amplitud de movimientos

8. **Indicar el título de la ventana sobre la que se va a utilizar**. Este paso es opcional. Mokey actúa sobre la ventana que se encuentre en primer plano, se introduzca el nombre de la ventana o no en el módulo mostrado en la figura A.1.8. En caso de introducir el título de la ventana en este módulo, la aplicación que vaya a ser utilizada se situará en primer plano, cuando el usuario se sitúe frente a la Kinect, de forma automática. En la figura A.1.9 se observa dónde se encuentra el título de ventana que se ha de introducir marcado en amarillo.

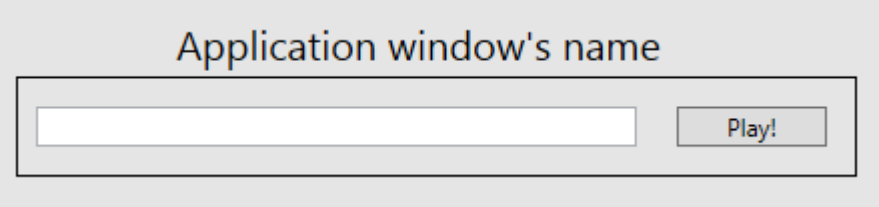

Fig. A.1.8 Módulo de selección de aplicación

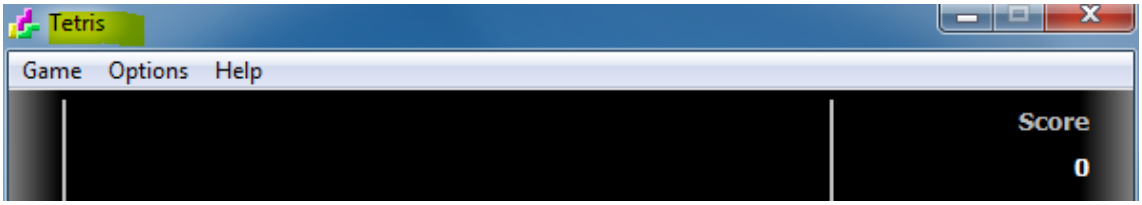

Fig. A.1.9 Título de ventana (Tetris)

9. **Pulsar el botón "Play!".** Una vez se haya completado la configuración, para activar el middleware se ha de hacer clic sobre el botón "Play!" de la figura A.1.8

10. **En caso de que se hayan seleccionado los movimientos que necesitan calibración** tras pulsar "Play!", aparecerá la ventana que se muestra en la figura A.1.10. Esta simplemente indica que el proceso de calibración va a ser iniciado. El usuario debe pulsar el botón "Aceptar" y dispondrá de 5 segundos para situarse en la posición de juego antes de que se haga la calibración.

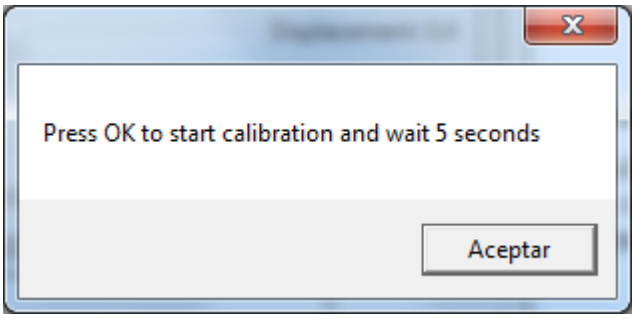

Fig. A.1.10 Ventana de aviso de calibración

11. **Tras situarse frente a la Kinect, Mokey ya puede ser utilizado**.

12. **Cerrar la interfaz gráfica cuando se quiera dejar de utilizar**. Cuando se desee cerrar el programa, se recomienda al usuario que se salga del rango de la cámara por un lateral. Se ha detectado en que raras ocasiones, cuando el usuario se sale del rango de la cámara acercándose a ella, se produce un bloqueo del programa. Tras esto, simplemente se ha de pulsar el botón de cerrar la interfaz como se cerraría cualquier ventana y se mostrará el aviso de la figura A.1.11 que confirma que el proceso y los hilos han sido correctamente finalizados.

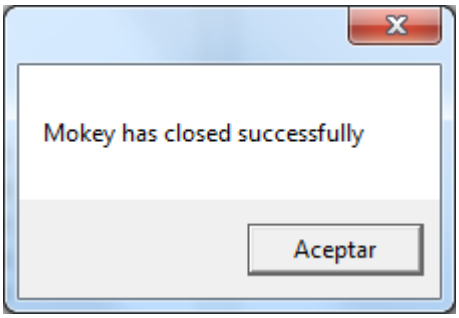

Fig. A.1.11 Venta de aviso de cierre

A 1.4 Usos de presets

Para facilitar el uso de la aplicación, Mokey permite al usuario guardar su configuración y poder cargarla en otro momento. Para ello se usa el módulo de presets que se observa en la figura A.1.12.

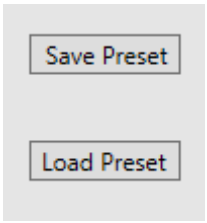

Fig. A.1.12 Módulo de presets

Los pasos a seguir para guardar la configuración son los siguientes:

1. **Realizar la configuración deseada** de Mokey mostrada en el apartado A 1.3.

2. **Pulsar el botón "Save"** que se observa en la figura A.1.12.

3. **Determinar el nombre del preset y su dirección de destino** en la ventana que se ve en la figura A.1.13.

4. **Tras esto, se obtiene un archivo con extensión .txt** en el destino seleccionado que contiene toda la información de la configuración. Bajo ningún concepto el usuario ha de editar el contenido de este archivo.

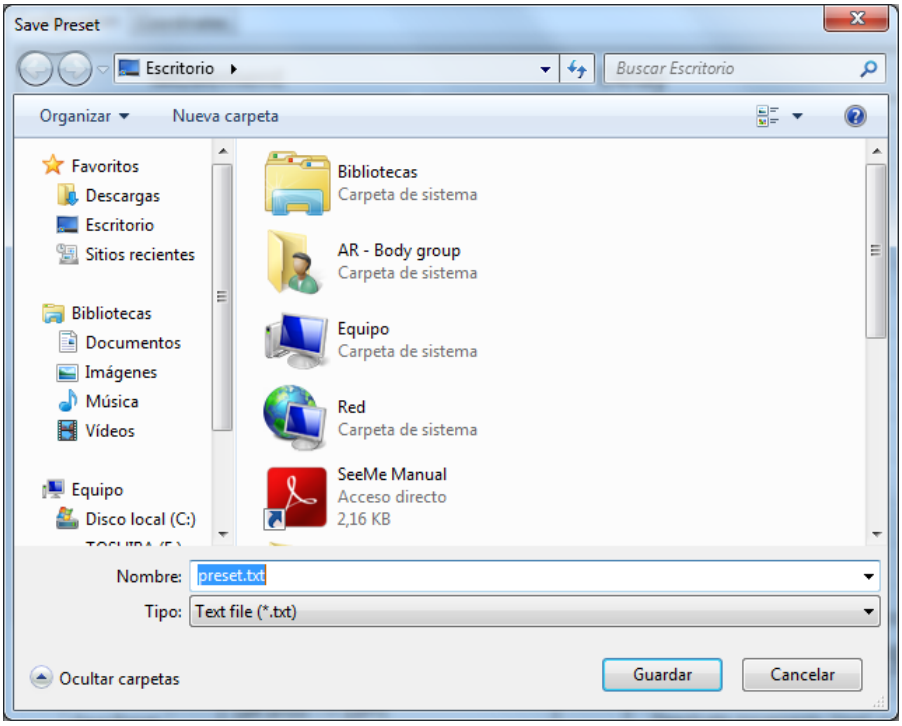

Fig. A.1.13 Ventana de guardado de presets

Los pasos a seguir para cargar la configuración son los siguientes:

1. **Pulsar el botón "Load"** que se observa en la figura A.1.12.

2. **Seleccionar el archivo .txt** que contiene la configuración en la ventana que se ve en la figura A.1.14.

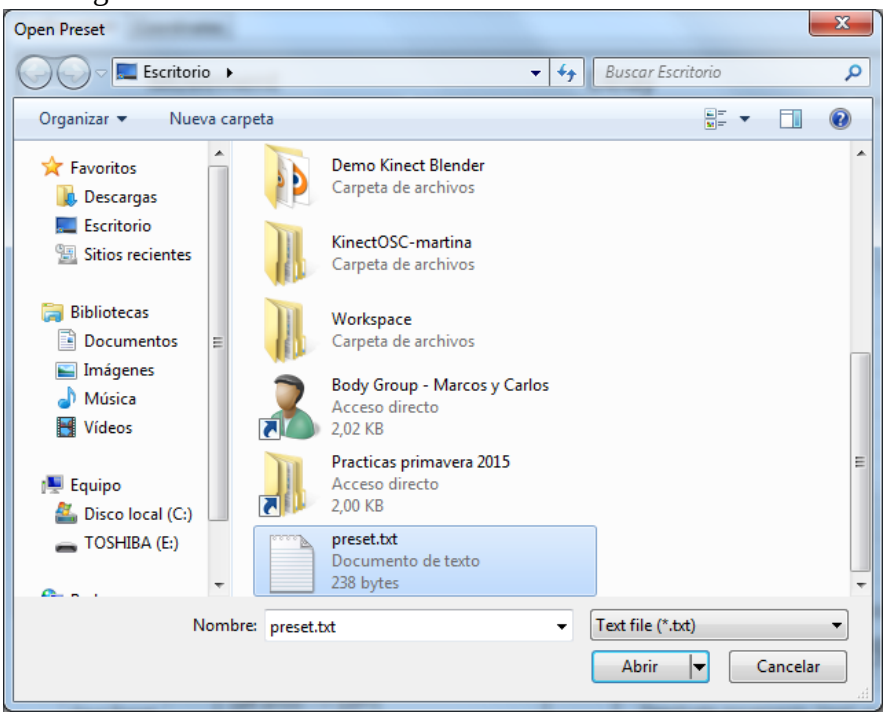

Fig. A.1.14 Ventana de carga de presets

3. Tras esto, **la interfaz gráfica es actualizada** con los valores presentes en el archivo.

4. Realizar los pasos del 9 en adelante, del apartado A 1.3.

A 1.5 Grabación de movimientos

Mokey permite al usuario la grabación de 4 movimientos propios, para complementar los 12 que ofrece. Para ello, ha de utilizar la pestaña "Record", donde se muestra la interfaz de grabación que se ve en la figura A.1.15.

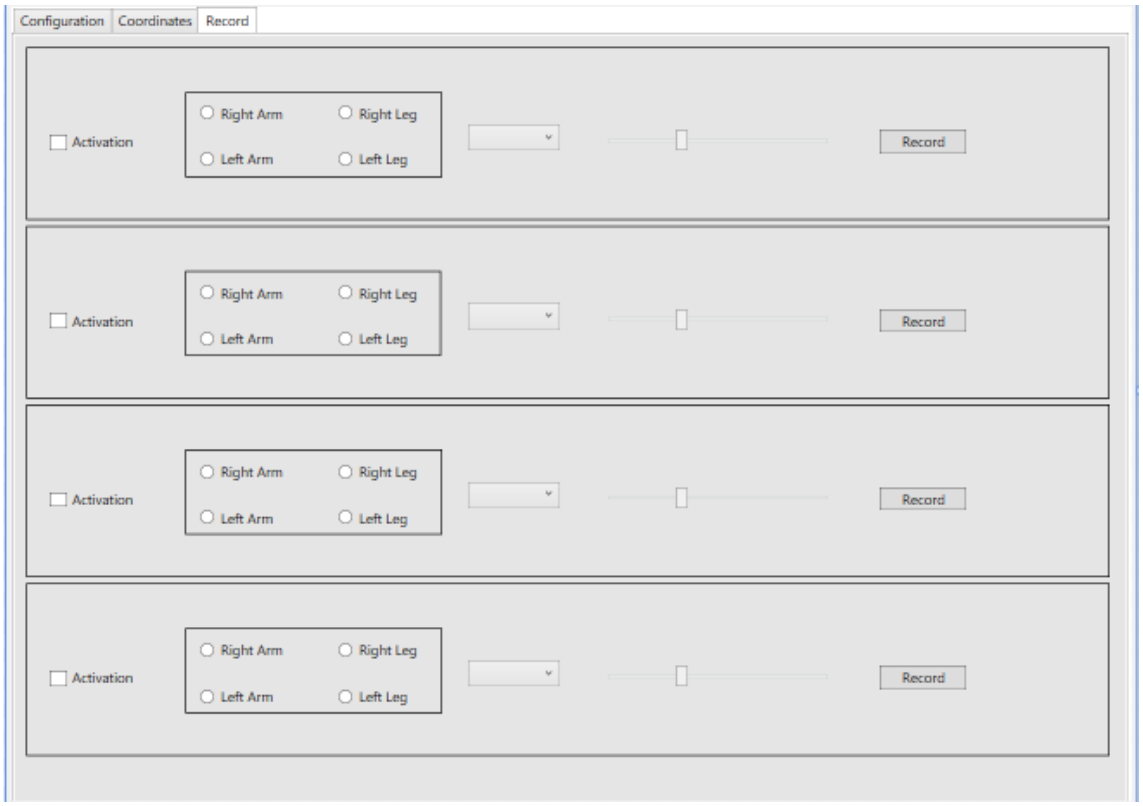

Fig. A.1.15 Interfaz gráfica de grabación de Mokey

Los pasos para grabar los movimientos son los siguientes:

1. **Seleccionar uno de los 4 módulos de grabación.** Es indispensable que cada grabación se haga en un módulo diferente, de lo contrario, se sobrescribirá el movimiento grabado.

2. **Seleccionar con qué miembro del cuerpo se quiere hacer el movimiento.** Seleccionar cuál de las piernas o brazos se va a ver implicado en el movimiento. En la figura A.1.16 se muestra dónde realizar la selección. Solo se puede elegir una de las opciones.

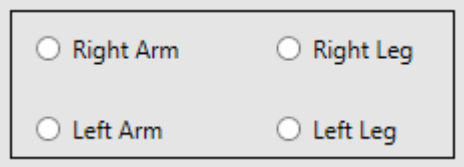

Fig. A.1.16 Módulo de selección del miembro corporal a grabar

3. **Colocarse frente a la Kinect realizando el movimiento grabar.** Únicamente hay que realizar la posición final del movimiento, ya que se graban las distancias del instante final. Por ejemplo, si quiere grabar el movimiento brazo derecho a la izquierda, mantenga el brazo fijo hacia la izquierda.

4. **Con la ayuda de otra persona, pulsar el botón "Record".** Una vez pulsado, la grabación es instantánea. El botón "Record" se puede observar a la derecha de cada uno de los cuatro módulos de grabación en la figura A.1.15.

5. **Se mostrará una ventana indicando que la grabación ha sido realizada.** En la figura A.1.17 se muestra dicha ventana. Es meramente informativa. Si los 3 valores mostrados para determinar la grabación son 0, esta no habrá sido realizada.

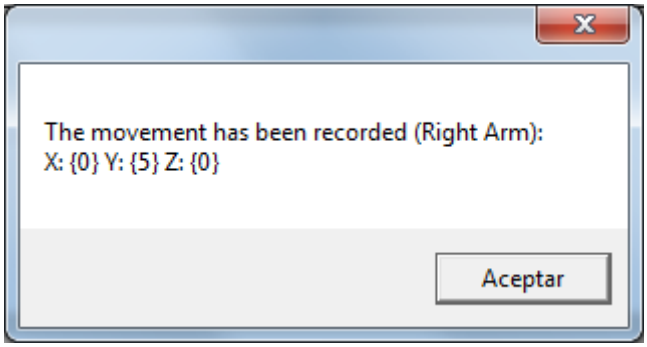

Fig. A.1.17 Ventana de confirmación de grabación.

6. **Configurar el retardo y tecla del movimiento.** En la figura A.1.15 se puede observar la caja para la activación del movimiento a la izquierda de cada módulo, el seleccionador de tecla y la barra deslizadora para elegir el retardo.

7. **Realizar los pasos del 9 en adelante, del apartado A 1.3.**

Advertencia: El usuario tiene que tener en cuenta que se pueden producir interferencias entre los diferentes movimientos grabados si estos se realizan con el mismo miembro corporal y en la misma dirección o similar.

ANEXO 2 Detalles del código de Mokey

En el presente anexo se detallan los aspectos más representativos del código del programa. El fin de este documento, es facilitar al lector la comprensión de los algoritmos planteados en este proyecto en C#. La información presente en este anexo está complementada con los comentarios que se encuentran en el propio código del programa.

A 2. 1 Importar librerías DLL

En la figura A.2.1 se puede observar el fragmento de código referente a la importación de dos funciones de la librería USER32.DLL.

```
25      public partial class MainWindow : Window {
26          //Importar librerias .DLL
27          [DllImport("USER32.DLL", CharSet = CharSet.Unicode)]
            1 referencia
28          public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);
29          [DllImport("USER32.DLL")]
            1 referencia
30          public static extern bool SetForegroundWindow(IntPtr hWnd);
```

Fig. A.2.1 Código para importar librerías DLL

Cómo se puede observar en dicho código, el proceso consta de dos partes, primero se indica la librería de la que se va a extraer la función (línea 27) y posteriormente se define dicha función (línea 28).

A 2.2 Definición de la Kinect

En la figura A.2.2 se muestra el código en el que se define la cámara Kinect.

```
286         private void cargar_Ventana(object sender, RoutedEventArgs e){
287             // Comprueba si la Kinect está conectada correctamente
288             if (KinectSensor.KinectSensors.Count == 0){
289                 MessageBox.Show("Kinect sensor is unplugged. Please, connect it");
290                 Application.Current.Shutdown();
291             }
292             // Obtiene los datos de la Kinect
293             myKinect = KinectSensor.KinectSensors[0];
294
295             // Activa y ejecuta la cámara Kinect. Activa la imagen de Esqueleto
296             try{
297                 myKinect.SkeletonStream.Enable();
298                 myKinect.Start();
299             }
300             catch{
301                 MessageBox.Show("Kinect initialise failed");
302                 Application.Current.Shutdown();
303             }
304
305             // Actualizar ventana constantemente...
306             myKinect.SkeletonFrameReady += new EventHandler<SkeletonFrameReadyEventArgs>(myKinect_SkeletonFrameReady);
307         }
```

Fig. A.2.2 Código para definir la Kinect

Se pueden observar un total de 4 partes en todo el proceso. En primer lugar de la línea 288 a la 290, se comprueba que la Kinect está conectada al ordenador. De no ser así, se informará al usuario de ello. Posteriormente se define un objeto donde se introducirán los datos de la Kinect como se ve en la línea 293. Posteriormente se inicializa dicho objeto, de manera que se puedan usar los datos de la Kinect en el programa, líneas 297 y 298. Finalmente como se ve en la línea 306, se indica que el proceso se repita constantemente

en el tiempo, para que los datos de esa variable que representa la Kinect, se mantengan actualizados con el movimiento actual del usuario.

A 2. 3 Creación del esqueleto

En la figura A.2.3 se muestran las líneas de código destinadas a crear un esqueleto operativo sobre el que se puedan actualizar los datos del esqueleto.

```
309         // Se establece el esqueleto por parte de Kinect
310         Brush skeletonBrush = new SolidColorBrush(Colors.Red);
            0 referencias
311         void addLine(Joint j1, Joint j2){
312             Line boneLine = new Line();
313             boneLine.Stroke = skeletonBrush;
314             boneLine.StrokeThickness = 5;
315 
316             DepthImagePoint j1P = myKinect.MapSkeletonPointToDepth(j1.Position, DepthImageFormat.Resolution640x480Fps30);
317             boneLine.X1 = j1P.X;
318             boneLine.Y1 = j1P.Y;
319 
320             DepthImagePoint j2P = myKinect.MapSkeletonPointToDepth(j2.Position, DepthImageFormat.Resolution640x480Fps30);
321             boneLine.X2 = j2P.X;
322             boneLine.Y2 = j2P.Y;
323         }
324 
```

Fig. A.2.3 Código para crear el esqueleto

A 2.4 Creación de los huesos

Cómo se ha visto en el proyecto, es esencial la definición de los puntos de interés para determinar si los movimientos se han cumplido. En la figura A.2.4 se muestran las líneas de código destinadas a ello. Debido a la gran extensión de código que supone definir todos los puntos, solo se muestran los fragmentos del pie derecho.

```
360                     // Pie Derecho
361                     Joint footRightJoint = skeleton.Joints[JointType.FootRight];
362 

377                         SkeletonPoint footRigthPosition = footRightJoint.Position;

441                      // Posicion Pie Derecho
442                      message5 = string.Format("RightFoot: X:{0:0.0} Y:{1:0.0} Z:{2:0.0}",
443                          pieDer_X = footRigthPosition.X,
444                          pieDer_Y = footRigthPosition.Y,
445                          pieDer_Z = footRigthPosition.Z);
446 
```

Fig. A.2.4 Código de creación de los huesos

Cómo se puede observar en el código, el proceso consta de tres partes, en primer lugar definir el punto del esqueleto del que se quiere obtener la información como se ven la línea 361. Posteriormente hay que generar una variable que almacene únicamente la posición de este punto como se ve en la línea 377. Esto se debe a que los puntos del esqueleto, tienen mucha más información que la posición. Posteriormente como se ve en la línea de la 442 a la 445, se extraen las coordenadas cartesianas de dicha posición.

A 2.5 Activación de los hilos.

En la figura A.2.5 se muestra como se activan los hilos de cada movimiento. Debido a la extensión del código únicamente se muestra el fragmento referente al movimiento 1.

```
513             if (chkmov1) {
514                 m1 = new Thread(new ThreadStart(Movimiento1));
515                 m1.Start();
516             }
```

Fig. A.2.5. Código de activación y creación de hilos

Cómo se observa en el código, en caso de que el usuario marque en la interfaz gráfica la caja del movimiento 1, chkmov1, en la línea 513, se creará un hilo llamado m1, con la rutina de trabajo Movimiento1, en la línea 514. Posteriormente en la línea 515, dicho hilo es activado.

A 2.6 Rutina de hilos

En la figura A.2.6 se muestra el código de la rutina asignada al movimiento 1.

```
593         public void Movimiento1() {
594             while(!parada){
595                 if (((brazoDer_Y - spine_Y) > s_mov1) && ((brazoDer_X - spine_X) < s_mov5)) {
596                     // Genera la pulsación del teclado virtual determinada en la interfaz
597                     System.Windows.Forms.SendKeys.SendWait(tecla1_aux);
598                 }
599                 Thread.Sleep(n_mov1); // Retardo
600             }
601         }
```

Fig. A.2.6 Código de rutina de movimiento 1

Cómo se puede observar en el código, en la línea 594 lo primero que se indica es que la rutina se repita de forma indefinida hasta que el programa sea parado. De otra forma, dicha rutina solo se realizaría una vez. Posteriormente se comprueba si la distancia entre los puntos de interés es la indicada para que se cumpla el movimiento. De ser así se manda el evento de teclado como se ve en la línea 597. En la línea 599 se muestra el retardo entre pulsaciones consecutivas que se elige en la interfaz.