Manual de Usuario del Sistema

Editor y reconocedor de gestos - 19-08-2011

Autores:

- *Fanaro Damián Ariel.*
- *Ibañez Rodrigo Sebastian.*

Profesor:

- *Dr. Soria Alvaro.*

Historial de cambios

Fecha	Versión	Cambio realizado	Responsable
19-08-2011	1.0	Creación del documento	Ibañez Rodrigo Sebastian

Indice

Overview

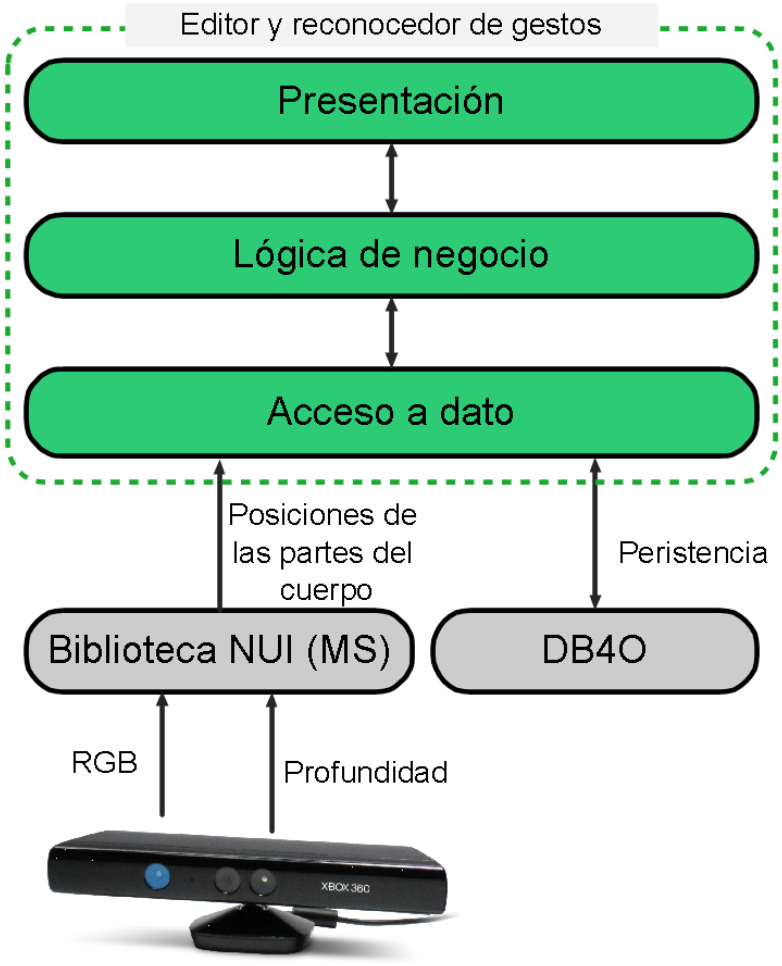
En este documento se pretende detallar el uso del "Editor y reconocedor de gestos". El cual surge en el contexto de la materia.

Sus características principales son:

- Detecta el cuerpo humano y hace un seguimiento de las partes del cuerpo de hasta dos personas simultáneamente. Para esto se utiliza el dispositivo Kinect de Xbox y el SDK de Microsoft que provee soporte para dicho dispositivo.
- Graba las posiciones del cuerpo en una secuencia para luego poder reproducirlas.
- Define estados del cuerpo humano, utilizando para esto posiciones relativas de las partes del cuerpo. Especificando por ejemplo: que la mano izquierda está sobre el hombro izquierdo, o la rodilla derecha sobre la cadera derecha.
- Define gestos del cuerpo humano como una secuencia temporal de estados.
- Reconoce y registra los estados y gestos creados.

Vista abstracta del sistema

La imagen muestra la arquitectura del sistema a grandes rasgos.



El recuadro punteado representa el editor y reconocedor de gestos. Dentro de este se puede ver la diferencia entre las tres capas.

La **capa de presentación**, que recibe eventos del usuario a través de la interfaz, y se encarga de manejar los mismos. Así como también de la parte visual de la aplicación.

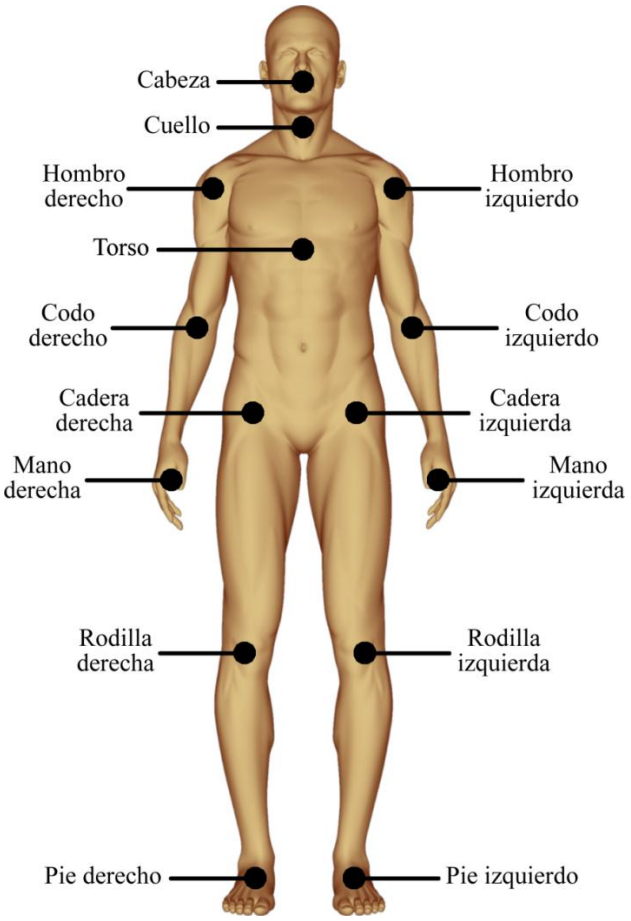
La **capa de lógica de negocio**, que contiene todas las estructuras necesarias que hacen al problema, como los estados los gestos, las animaciones.

La **capa de acceso a datos**, en la cual simplemente hay lógica que lleva y trae información entre la capa de negocio y los sistemas externos donde los datos se almacenan o producen. (Producen en el sentido que por ésta vía se accede al dispositivo kinect el cual no posee datos almacenados sino que los emite a medida que se van capturando).

Conceptos

Aquí se detallan conceptos y decisiones de interés para el usuario que se tomaron en el desarrollo de la aplicación.

- Se modela el esqueleto humano utilizando una estructura propia debido a que esto da más libertad a la hora de realizar posibles cambios futuros. La estructura utilizada se llama **HumanSkeleton**, y tiene una lista de las partes del cuerpo **BodyPart**. Cada BodyPart se identifica por el nombre de la parte del cuerpo y contiene además la posición en el espacio donde está ubicada.
  Las partes del cuerpo que se utilizan en la aplicación son las 15, que se muestran a continuación:



- Se creó una **AbstractMiddle**, para que mantenga actualizadas las posiciones de las partes del cuerpo del **HumanSkeleton**. Ese **AbstractMiddle** se especializó en **KinectMiddleMS** si actualiza las posiciones del cuerpo desde el dispositivo kinect utilizando la API de Microsoft. O **SkeletonReaderMiddle**, si actualiza las posiciones desde un archivo.

- Para la creación de estados se implementó un filtro **AbstractState**, el cual dado un **HumanSkeleton**, devuelve verdadero o falso si se cumple con dicho filtro. Es importante destacar, que se optó por crear estados teniendo en cuenta las posiciones relativas del cuerpo de forma tal que la distancia de la persona al dispositivo, o el tamaño de la persona no afecten con el cumplimiento o no de dichos estados.
  Los criterios de filtrado son los siguientes:
  - AndState, verifica que se cumplan los dos estados que lo componen.
  - AngleState, verifica que se cumpla un ángulo entre tres partes del cuerpo.
  - CenterStateX, verifica que la primer parte del cuerpo este centrada en el eje x con respecto a la segunda.
  - CenterStateY, verifica que la primer parte del cuerpo este centrada en el eje y con respecto a la segunda.
  - CenterStateZ, verifica que la primer parte del cuerpo este centrada en el eje z con respecto a la segunda.
  - DistanceState, verifica que la distancia entre dos partes del cuerpo sea menor que un valor.
  - InvalidState, devuelve siempre falso.
  - LeftState, verifica que la primer parte del cuerpo este a la izquierda de la segunda.
  - OrState, verifica que se cumpla alguno de los dos estados que lo componen.
  - OverState, verifica que la primer parte del cuerpo este arriba de la segunda.
  - PositionState, verifica que la parte del cuerpo este en una posición fija respecto del centro del cuerpo.
  - RightState, verifica que la primer parte del cuerpo este a la derecha de la segunda.
  - UnderState, verifica que la primer parte del cuerpo este debajo de la segunda.

- Para la creación de gestos se implementó un nuevo filtro **AbstractGesture**, el cual consta de una lista de estados y tiempo máximo asociado al cumplimiento de dicho estado. Ese **AbstractGesture** se especializó en **SimpleGesture** encargado de avanzar la lista si el estado es cumplido para verificar el cumplimiento del gesto. O reiniciar el gesto, en caso de que el tiempo máximo asociado al estado se haya agotado.

Funciones básicas

A continuación se explican los códigos necesarios para utilizar la herramienta en otra aplicación. Es importante antes que nada, explicar el ciclo a seguir para que todo funcione correctamente:

- Crear un HumanSkeleton.
- Inicializar el HumanSkeleton. Con las 15 partes del cuerpo.
- Crear un Middle.
- Agrega al Middle un HumanSkeleton o dos según los usuarios que quieras reconocer.
- Por siempre:
  - Actualizar el Middle.
  - Utilizar el HumanSkeleton para lo que se quiera.

Creando el HumanSkeleton

Lo primero que hay que hacer es crear un HumanSkeleton e inicializarlo para esto lo hacemos de la siguiente manera:

```
//Declaración
private HumanSkeleton mySkeleton1;
private HumanSkeleton mySkeleton2;
//Esto es porque se va a reconocer 2 usuarios

//inicialización
mySkeleton1 = new HumanSkeleton();
mySkeleton2 = new HumanSkeleton();

//Carga las 15 partes del cuerpo para ambos esqueletos
mySkeleton1.init();
mySkeleton2.init();

myMiddle = new KinectMiddleONI(mySkeleton);
```

Inicializando dispositivo

Una vez cargado el esqueleto debemos inicializar un dispositivo de entrada de datos, ya sea "Kinect" o un archivo donde se tienen cargados los movimientos. Para esto se utiliza un Middle.

El Middle necesita uno o más HumanSkeleton donde va a cargar los puntos detectados de cada parte del cuerpo.

Para inicializar el dispositivo lo hacemos de la siguiente manera:

```
//Declaración
private AbstractMiddle myMiddle;

//inicialización
//Para iniciar el dispositivo kinect.
myMiddle = new KinectMiddleMS();
myMiddle.addUser(mySkeleton1);
myMiddle.addUser(mySkeleton2);

o

//Para simular el inicio del dispositivo desde un archivo.
myMiddle = new SkeletonReaderMiddle();
myMiddle.addUser(mySkeleton1, path1);
myMiddle.addUser(mySkeleton2, path2);
```

Actualizando el modelo

Una vez inicializado el dispositivo, el Middle, provee el método update(), el cual se encarga de actualizar lo que está viendo el dispositivo Kinect o avanzar el archivo y copiar la nueva información en los HumanSkeleton que le agregamos. Además provee un método getState(), el cual nos indica el estado del dispositivo.

Este método puede dar 3 estados del dispositivo que son:

BUSCANDO: el Middle no está listo, está buscando a alguien.

CALIBRANDO: el Middle no está listo, encontró a alguien pero necesita calibrarlo.

DETECTADO: el Middle está listo y está actualizando las nuevas posiciones del HumanSkeleton.

No necesita descripción pero se haría de la siguiente manera:

```
    while (true)
    {
        //Actualizar los HumanSkeleton
        myMiddle.update();

        //Utilizar la información actualizada de los HumanSkeleton ya sea para
dibujar, guardar las posiciones, o detectar estados y gestos.
        ...
    }
```

Dibujar el modelo

Una vez que el dispositivo está listo y los valores del HumanSkeleton están actualizados, hay que dibujarlo. Lo hacemos de la siguiente manera

```
    //Dibujando
    if (myMiddle.getState(0).Equals(myMiddle.DETECTADO))
    {
        foreach (DictionaryEntry dE in mySkeleton1.getBodyParts())
               dibujar((HumanBodyPart)dE.Value);
    }
    if (myMiddle.getState(1).Equals(myMiddle.DETECTADO))
    {
        foreach (DictionaryEntry dE in mySkeleton1.getBodyParts())
               dibujar((HumanBodyPart)dE.Value);
    }
    //0 y 1 corresponde al número de usuario, posicional de acuerdo al orden en que
fueron agregados al Middle
```

El **update()** y el dibujar se deben llamar de forma alternada para simular el movimiento.

La función dibujar no se explica porque depende de cada plataforma que se utilice. Pero cada **HumanBodyPart** contiene `una posición (X, Y, Z).`

Guardando el modelo

Para poder guardar el esqueleto lo que se debe hacer es inicializar un **SkeletonSaveMiddle**, que hereda de Middle. Luego hay que agregarle los HumanSkeleton que queremos guardar con el path donde se almacenará cada uno. Y luego de actualizar el esqueleto, y que el dispositivo esté listo utilizar el método saveState(), el cual va a guardar el estado actual del esqueleto. Lo

```
//Declaración
private AbstractMiddle myMiddle;
private HumanSkeleton mySkeleton;
private SkeletonSaveMiddle mySkeletonSave;

//inicialización
myMiddle = new KinectMiddleMS();
myMiddle.addUser(mySkeleton);

//Para iniciar el grabador del HumanSkeleton
mySkeletonSave = new SkeletonSaveMiddle();
mySkeletonSave.addUser(mySkeleton, path);
```

hacemos de la siguiente manera

El **update()** y el dibujar se deben llamar de forma alternada para simular el movimiento.

La función dibujar no se explica porque depende de cada plataforma que se utilice. Pero cada **HumanBodyPart** contiene `una posición (X, Y, Z).`

Guardando el esqueleto

Para poder guardar el esqueleto lo que se debe hacer es inicializar un SkeletonSave, pasándole como parámetro el HumanSkeleton. Y luego de actualizar el esqueleto, y que el dispositivo esté listo utilizar el método saveState(), el cual va a guardar el estado actual del esqueleto. Lo hacemos de la siguiente manera:

```
//Declaración
private SkeletonSave mySkeletonSave;

//inicialización
//Para iniciar el grabador del HumanSkeleton
mySkeletonSave = new SkeletonSave(mySkeleton, "grabacion.txt");


//Guardando el esqueleto
            if (myMiddle.getState().Equals(myMiddle.DETECTADO))
            {
                mySkeletonSave.saveState();
            }

//Cerrando el SkeletonSave
             if (mySkeletonSave != null)
                mySkeletonSave.cerrarArchivo();
```

El update() y el guardar esqueleto se deben llamar de forma alternada para guardar de forma continua lo que se está capturando.