Universidad de las Ciencias Informáticas
Vertex, Entornos Interactivos 3D, Facultad 4

Plugin de Unity para el acceso a los sensores de un dispositivo con Android.

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

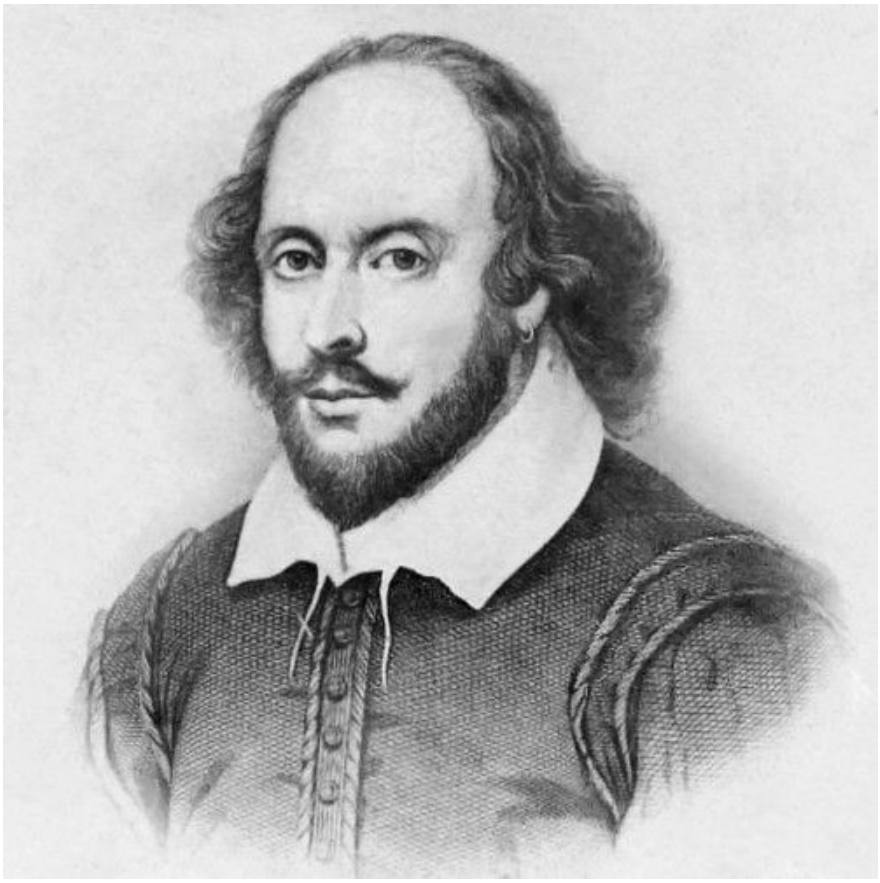
Autor: Andro Rodríguez Martínez

Tutores: Ing. José Lozano Hernández

MSc. Andy Hernández Paez

Ing. Angel Ulise Tabares González

La Habana, 2018



Nuestras dudas son traidoras, y por ellas perdemos el bien que con frecuencia pudimos ganar, por miedo a intentarlo.
William Shakespeare

Dedicatoria

A lo más grande que tengo en la vida, mi madre María Victoria, por todo su apoyo, amor y cariño durante estos duros 23 años, por ser mi luz en el camino de la vida, porque nunca se rindió conmigo ni con mis hermanos y siempre nos mostró el lado bueno de las cosas a pesar de los duros tropiezos y por luchar día a día para que nunca me faltara nada. A mi padre Arnaldo Rodríguez por todos sus consejos, por ser mi amigo en todo momento, por mostrarme cómo ver la vida y por darme su apoyo incondicional para que no me rindiera ante los obstáculos de la vida. A mi abuelita Palmira, mi tía Onelia, mi prima Maydín y mis hermanos por brindarme su apoyo en todo momento. En fin, a toda mi familia por siempre estar ahí para mí.

Agradecimientos

Hoy si he logrado alcanzar una de las metas más importantes en mi vida e iniciarme como profesional ha sido gracias a muchas de las personas presentes, por eso me gustaría agradecerle a:
Mi novia Marlen, por soportarme tanto tiempo, por los lindos momentos que hemos pasado juntos y por apoyarme siempre. Gracias a mi tutor José Lozano por toda la ayuda y apoyo que me ha brindado durante todo este tiempo y por la posibilidad de realizar este trabajo, a mis co-tutores Andy Hernández por apoyarme con el documento y Ulises Tabares por despertar mi matemática, a Rubén, Alina y el Guille por presionarme para que llevara a cabo este proyecto y a Zaida por ser más que una profesora guía.
Agradecerles a mis hermanos, aunque no de sangre Yansel, Omar, Randy, Lidice e Ivett, por toda su amistad, por estar en mis peores momentos y en los mejores, los quiero. A el combo de los malos tipos por haber luchado junto a mí hombro con hombro y por su amistad durante estos 5 años. A las bellezas latinas de mi aula por darme momentos de alegría interminables, a los riquitis del 8 por contar siempre conmigo para todo, a mi combo de la facultad 3, a las muchachitas de redes, a las fiesteras de mi facultad, en fin, quiero agradecerles a todos por su apoyo incondicional ante todo tipo de situaciones, por su cariño, su amistad, sus consejos, por todo.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Andro Rodríguez Martínez
Autor

Ing. José Lozano Hernández
Tutor

MSc. Andy Hernández Paez
Tutor

Ing. Angel Ulise Tabares González
Tutor

Resumen

Con el transcurso del tiempo la tecnología móvil ha evolucionado y se le han incorporado nuevas funcionalidades con el objetivo de alcanzar un mayor desarrollo promocional. El Centro de Entornos Interactivos 3D (VERTEX) ha sido participe en la producción de nuevas aplicaciones móviles. La elaboración de la presente investigación surge a través de una propuesta planteada por el centro VERTEX, de construir un plugin en Unity que acceda a los sensores disponibles en un dispositivo móvil, debido a que esta herramienta no tiene acceso a todos ellos. Este trabajo se enfoca en crear un método capaz de corregir la información del sensor de orientación, con el objetivo de que los dispositivos móviles de gama baja sean capaces de obtener la orientación espacial. Para orientar el desarrollo del trabajo, desde el análisis hasta las pruebas realizadas, se siguió detalladamente las fases que propone la metodología Programación Extrema (XP). Se utilizaron Android Studio y Unity como herramientas de trabajo. La biblioteca diseñada, accede a los sensores de un dispositivo móvil y se implementó un método capaz de obtener la orientación espacial. Para realizar las pruebas, se comparó la diferencia angular entre las orientaciones obtenidas por el sensor giroscopio y el método implementado, los resultados arrojaron una diferencia media de 7.282 grados en el eje X, 8.836 grados para el eje Y y 5.069 grados para el eje Z. Las pruebas realizadas, demuestran que el uso de la solución desarrollada mejora considerablemente la orientación espacial en los dispositivos móviles de gama baja.

Palabras clave: Android, orientación, plugin, sensores, Unity.

Índice general

Índice de figuras

Índice de tablas

Introducción

La tecnología avanza a pasos de gigante, los teléfonos móviles y la industria de videojuegos evolucionan en la misma medida. El sistema operativo Android, desarrollado por Google y basado en Linux, es una plataforma que aún está en proceso de evolución y queda reflejado mediante las versiones que periódicamente se lanzan al mercado. Android ha sido la elección de muchas industrias que fabrican teléfonos inteligentes por su versatilidad y por ser de código abierto. Android fue creado por la empresa Android Inc., una firma que adquirió Google en el 2005 (Ulmeher, 2015).
La industria de videojuegos se puede considerar consagrada en todo el mundo, posee un mercado que va en ascenso, y junto a ello, los desarrolladores de videojuegos. El centro VERTEX ubicado en la Universidad de las Ciencias Informáticas (UCI) es vanguardia en esta industria en Cuba. La mayoría de los productos se desarrollan con la herramienta Unity, debido a que facilita el desarrollo en diferentes plataformas como escritorio, consolas, móviles y web. El desarrollo de los dispositivos móviles está acompañado de la incorporación de componentes electrónicos y sensores, los cuales brindan nuevas funcionalidades y experiencias de usuario. Durante la concepción de videojuegos, se tiene en cuenta el uso de estos sensores para enriquecer la interacción con el usuario. Unity brinda una interfaz para el acceso a sensores, tanto en Android como iOs. Debido a temas de licencia de software, el centro se enfocó en desarrollar aplicaciones para Android.

Durante el desarrollo de una serie de aplicaciones para Android usando Unity, surge la siguiente **situación problemática**:

- Unity brinda una interfaz muy básica para acceder a la información de los sensores de un dispositivo.
- La orientación espacial se puede obtener solo por medio del sensor giroscopio, y muchos dispositivos no poseen este sensor. Debido a que el sensor giroscopio no es un componente que poseen todos los dispositivos móviles, se puede utilizar el sensor orientación, pero no está implementado en Unity el acceso a él.
- La información del sensor de orientación en Android, no tiene un formato completamente compatible para utilizarlo como rotación en Unity, y se deben realizar ajustes.

Teniendo en cuenta la situación problemática descrita anteriormente, se expone el siguiente **problema de investigación:** ¿Cómo acceder a los sensores disponibles en un dispositivo con Android utilizando Unity y corregir además la información de orientación? Mediante el cual se escoge como **objeto de estudio** el acceso a los sensores de un dispositivo y corrección de la información de orientación espacial, enmarcán-

dose en el **campo de acción**, corrección de la información de orientación espacial brindada por el sensor de orientación de Android, declarándose como **objetivo general:** Desarrollar un plugin de Unity para el acceso a los sensores de un dispositivo con Android, que incluya un método de corrección para el sensor de orientación. Para cumplir con el objetivo general, se definen las siguientes **tareas investigativas**:

- Elaboración del marco teórico sobre los sensores disponibles en un dispositivo móvil con Android.
- Caracterización del entorno de desarrollo de Unity para la implementación de un plugin que acceda a los sensores de Android.
- Selección de la metodología, herramientas y tecnología para el desarrollo de un plugin en Unity que acceda a los sensores disponibles en Android.
- Análisis, diseño e implementación de la solución propuesta.
- Validación de la propuesta de solución mediante las pruebas definidas en la investigación.

Para llevar a cabo la elaboración de este trabajo se tuvieron presente los siguientes métodos científicos de investigación:

Métodos empíricos:

- Observación: Se utiliza para analizar los resultados de la orientación y observar cómo influye en el sensor de Android.

- Pruebas: Se emplean una secuencia de análisis a los algoritmos para determinar si cumplen con los resultados que se desean.

Métodos teóricos:

- Analítico Sintético: Se elaboran documentos basados en los sensores que brinda el sistema operativo Android y recopilar toda la información necesaria acerca de cómo resolver el problema del sensor orientación.

- Inductivo Deductivo: Una vez analizada toda la información relacionada con las matrices de la orientación espacial, se pueden adquirir ideas para tratar el problema en cuestión.

- Histórico Lógico: Se estudian los conceptos, evolución y desarrollo del sistema operativo Android y sus componentes más importantes.

Método estadístico:

Este método se utiliza para el análisis de los resultados obtenidos en las iteraciones de las pruebas desarrolladas, y el nivel de satisfacción con la propuesta de solución.

La presente tesis está estructurada por tres capítulos, los cuales se presentan a continuación:

Capítulo 1: Fundamentación teórica. En el desarrollo de este capítulo se definen las bases teóricas del sistema operativo Android y se da a conocer cómo se encuentra estructurado, además se presentan los principales sensores que forman parte del sistema operativo Android. También se refleja el concepto de plugin y sus características fundamentales. Se realiza un breve estudio de la orientación espacial que se utilizará en la solución.

Capítulo 2: Características y diseño del sistema. En este capítulo se propone una solución al problema, donde se explican los pasos a seguir para crear una biblioteca que contenga los sensores principales de Android. También se realiza un método de corrección de la información del sensor de orientación, teniendo

en cuenta las técnicas y algoritmos utilizados. Además, se hace una breve explicación de cómo se trabajó en Unity para la creación de un plugin para Android que implemente los sensores utilizados en la biblioteca. Se realiza la ingeniería del sistema y se define el diseño.

Capítulo 3: Implementación y pruebas. En este capítulo es donde se define la fase de implementación, y se analizan las tareas de ingenierías correspondientes a cada Historia de Usuarios (HU). Además, se han realizado las pruebas de aceptación para comprobar si el desarrollo de las funcionalidades implementadas es el correcto.

CAPÍTULO 1

Fundamentación teórica

1.1. Introducción al capítulo

En el presente capítulo se tratarán una serie de componentes que integran el sistema operativo Android. Se definen varios conceptos que serán utilizados durante el trabajo. Además, se expondrán los tipos de sensores que existen y cómo funcionan. Entre los componentes a caracterizar se encuentra el plugin y la plataforma Unity. Se explica en qué consiste la orientación espacial de un dispositivo móvil, utilizando los conceptos de rotación, *quaternion* y la orientación en los dispositivos móviles.

1.2. Sistema Android

Android es un sistema operativo y una plataforma de software basado en Linux para teléfonos móviles. Android se utiliza además en *tablets*, *netbooks*, reproductores de música e incluso ordenadores.
El sistema operativo Android surge a través de la empresa Android Inc., teniendo como principales desarrolladores a Andy Rubin, Rich Miner, Nick Sears y Chris White. El objetivo principal era convertir Android en un sistema para cámaras, el cual no funcionó debido a que en aquel entonces no existía suficiente desarrollo en cuanto a este mercado, entonces, sus desarrolladores centraron su enfoque en dispositivos móviles. Google pasa a adquirir Android Inc., en 2005, unos años más tarde se crea la Alianza Abierta de Teléfonos (OHA, por sus siglas en inglés) conformada por 48 empresas dedicadas a la comercialización, la telefonía y fabricantes de electrónicos (Álvaro Borrego, 2012). Android es uno de los Sistemas Operativos móviles más utilizados en el mundo, por lo que existe un gran mercado para los desarrolladores de aplicaciones.

1.2.1. Estructura de Android

Android tiene acceso a todos los recursos del Kernel de Linux, y su arquitectura se conforma de la siguiente manera:

- **Aplicaciones:** son las distintas aplicaciones y servicios que se pueden ver en el dispositivo Android. En este nivel es donde se desarrollan las aplicaciones.
- **Marco de trabajo de aplicaciones:** Los desarrolladores tienen acceso a este marco por medio de Interfaz de Programación de Aplicaciones (APIs, por sus siglas en inglés) que a través de ellas, se acceden a funcionalidades específicas implementadas en el Sistema Operativo.
- **Bibliotecas:** Conjunto de bibliotecas que permiten utilizar las distintas características extras que no se incluyen en Java ni en el núcleo. Estas bibliotecas están escritas en C/C++, algunas de ellas son:
  - Gestor de superficies: Es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.
  - Biblioteca Open GL/ES y SGL: Representan las bibliotecas gráficas y, por tanto, sustentan la capacidad gráfica de Android. OpenGL/ES maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el hardware encargado de proporcionar gráficos 3D. Por otro lado, SGL proporciona gráficos en 2D, por lo que será la biblioteca más habitualmente utilizada por la mayoría de las aplicaciones. Una característica importante de la capacidad gráfica de Android es que es posible desarrollar aplicaciones que combinen gráficos en 3 y 2D.
  - Biblioteca FreeType: Permite trabajar de forma rápida y sencilla con distintos tipos de fuentes.
  - Biblioteca SSL: Posibilita la utilización de dicho protocolo para establecer comunicaciones seguras.
  - Biblioteca SQLite: Creación y gestión de bases de datos relacionales.
  - Biblioteca WebKit: Proporciona un motor para las aplicaciones de tipo navegador y forma el núcleo del actual navegador incluido por defecto en la plataforma Android.
- **Runtime de Android:** Al mismo nivel que las bibliotecas de Android se sitúa el entorno de ejecución, incluye el conjunto de bibliotecas que proporciona la mayor parte de la funcionalidad disponible en el núcleo de las bibliotecas de Java, y la máquina virtual Dalvik.
- **Linux Kernel:** Sección basada en el Kernel o núcleo de la versión 2.6 de Linux, contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las bibliotecas de control o drivers necesarios dentro de este Kernel de Linux embebido en el propio Android. Éste actúa como una capa de abstracción entre el hardware y el software.

En la figura 1.1 se puede visualizar como se encuentra compuesta la arquitectura del sistema operativo Android.

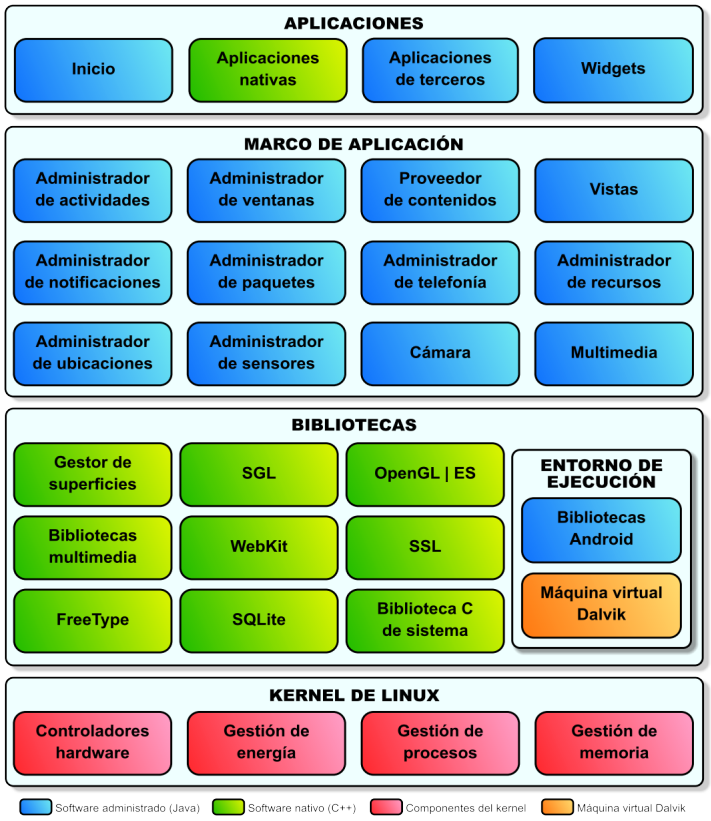


Figura 1.1. Arquitectura de Android

1.3. Sensores

A medida que la ciencia avanza, la tecnología le sigue los pasos y con ello los dispositivos móviles, a los cuales se les incluyen nuevos componentes y funcionalidades para mejorar su rendimiento. Un sensor es un dispositivo que está capacitado para detectar acciones o estímulos externos y responder en consecuencia. Es decir, permiten captar la información del medio físico que los rodea. Se encargan de medir las magnitudes físicas y transformarlas en señales eléctricas capaces de ser entendidas por un microcontrolador capaz de detectar estas magnitudes (Ana Gardey, 2010). A continuación, aparecen los sensores más utilizados en la actualidad, tales como los de posición, aceleración, proximidad, de luz entre otros.

1.3.1. Acelerómetro

El acelerómetro es un componente electrónico similar a un chip pequeño debido a que funciona con nanotecnología, fabricado en silicio donde su funcionamiento se basa fundamentalmente en la física, en medir aceleraciones por la fuerza de gravedad y cambios de movimiento. El acelerómetro es tan sensible que registra fuerzas diminutas, ya sea por la gravedad o el entorno. Esto hace que los datos generados por el acelerómetro contengan “ruido”. Su funcionamiento está centralizado en un condensador, un dispositivo

que se encuentra integrado por dos placas metálicas fijas una en frente de la otra y por un material dieléctrico que puede ser poliéster o papel (Juan de Urraza, 2009).
El acelerómetro contiene tres tubos pequeños en su interior, ubicados en la parte superior, cuenta con un muelle y en la otra punta existe una bola que funciona como masa. Esto significa que, si hay tres tubos simulando los tres ejes de coordenadas tridimensionales, al mover este conjunto, la bola que hace de masa se empezará a mover en los tubos y de esta manera se obtendrá la posición en la que se encuentra el dispositivo, en la figura 1.2 se muestra como se orientan los ejes de coordenadas de un acelerómetro en un teléfono (Rivera, 2012).

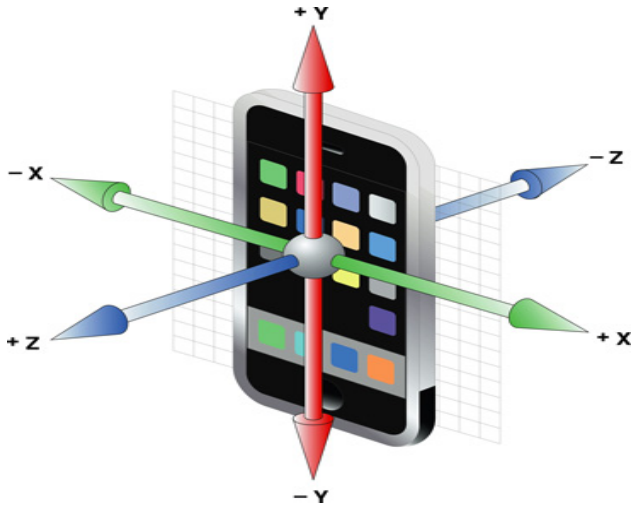


Figura 1.2. Acelerómetro

1.3.2. Campo magnético

Este sensor es un componente electrónico encargado de medir y cuantificar la cantidad de fuerza magnética que existe en un objeto, su resistencia eléctrica cambia según el ángulo de incidencia de un campo magnético y, al igual que el acelerómetro se construye sobre tres ejes. Mayormente su funcionalidad en los dispositivos móviles es similar a la de una brújula, captando el polo norte magnético (Gallego, 2015).

1.3.3. Giroscopio

El giroscopio se utiliza para adquirir la posición de los dispositivos de una forma más eficiente. Este sensor es un componente mecánico que permite mantener, medir y cambiar la orientación de un dispositivo. El giroscopio se basa principalmente en el momento angular, detecta la rotación a la que se somete el dispositivo y una de sus utilidades más comunes es medir la velocidad de giro (ibíd.).
Tomando como ejemplo un objeto en forma de cilindro y se atraviesa en un disco, el cual se va a hacer girar libremente sin que roce con el objeto, se aplica una fuerza compensatoria con las dos manos en cada lado

del palo con el disco girando y al instante tiende a estabilizarse permitiendo a quien sostiene el palo saber la posición, en la figura 1.3 se puede observar un giroscopio rústico.

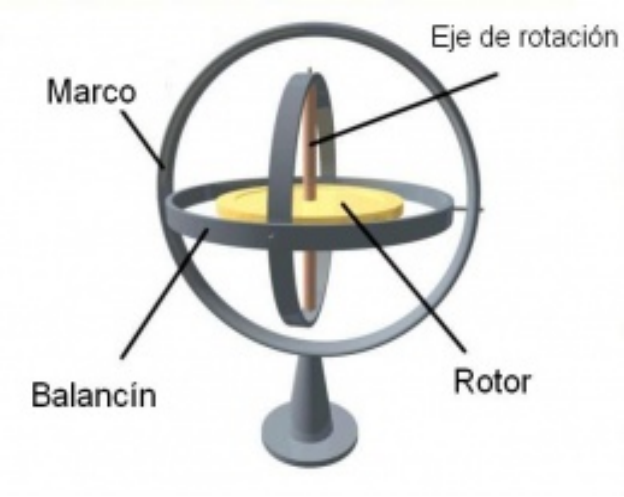


Figura 1.3. Sensor giroscopio

1.3.4. Sensor de Orientación

Se basa, como ya lo dice su nombre, en detectar la orientación del dispositivo y uno de sus empleos es en la realidad aumentada. Funciona combinando el acelerómetro y el sensor campo magnético, donde, el acelerómetro le brinda información acerca de cómo obtener un vector al centro de masa de la tierra y el campo magnético le facilita un vector al polo norte magnético, al mezclar esta información se puede conocer hacia donde apunta el dispositivo móvil en sus tres ejes, como se muestra en la figura 1.4 (Gironés, 2012).

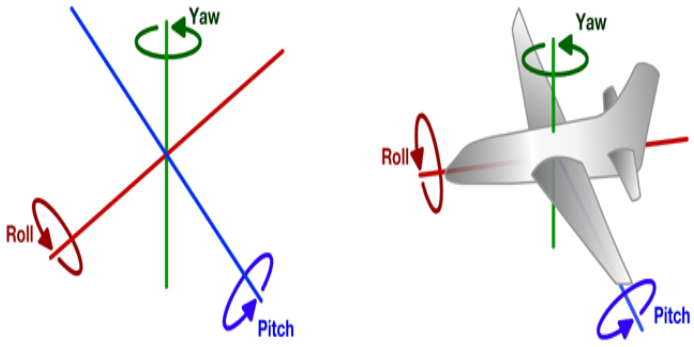


Figura 1.4. Sensor de Orientación

1.3.5. Sensor de proximidad

Los sensores de proximidad se centran en un LED infrarrojo y en un receptor infrarrojo, que emiten una luz infrarroja y una vez devuelta al receptor, detecta si existe algún objeto cerca del dispositivo, en caso de ser una llamada el receptor detecta la oreja de la persona. Una de las formas de empleo más utilizadas en la actualidad es a la hora de desbloquear un teléfono, con solo pasar el dedo cerca del sensor de proximidad, este se encuentra ubicado en el costado de la cámara frontal como aparece reflejado en la figura 1.5 (Gallego, 2015).

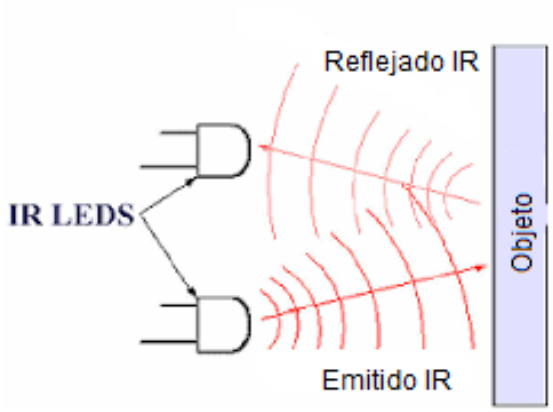


Figura 1.5. Funcionamiento del sensor de proximidad

1.3.6. Sensor de Luz

Este sensor brinda información sobre la cantidad de luz que está incidiendo en el dispositivo, su función es detectar qué intensidad tiene la luz del ambiente para ajustar el brillo de la pantalla. Debido a su diseño, los cambios de brillo se realizan de modo gradual a fin de evitar el parpadeo. Cada vez que sea utilizado en una región que se encuentra muy iluminada, la pantalla incrementará su brillo, por el contrario, opacará la pantalla cuando sea utilizada en áreas de menor claridad. Este sensor es de gran importancia para el ahorro de batería en los dispositivos móviles.
Existen otros modelos que contienen el sensor RGB, este detecta varios colores (en específico el rojo, azul, verde y el blanco) en la luz para modificar el brillo en la pantalla de una manera eficiente y así mejorar la nitidez (ibíd.).

1.3.7. Barómetro

El barómetro permite medir la presión ambiental, su uso se refleja en los teléfonos inteligentes de gama alta. Los datos que mide el dispositivo son de utilidad para determinar en qué nivel por encima del mar se encuentra el teléfono, una vez obtenidos estos datos se mejora la precisión del GPS. Los primeros teléfonos

inteligentes que le dieron utilidad fueron el Samsung Galaxy Nexus y el Motorola XOOM. Este sensor no tuvo demasiado éxito y se prescindió de él en favor de otros como los sensores de humedad (Gallego, 2015).

1.3.8. Sensor de humedad ambiente

Se basa en medir la cantidad de agua que hay en el ambiente. Los sensores de humedad pueden ser capacitivos o resistivos y también pueden medir la humedad relativa, absoluta o punto de rocío. Llevándolo a la práctica el Samsung Galaxy S4 es uno de los pocos dispositivos que contiene este sensor que permite controlar la humedad para, por ejemplo, saber si el usuario está o no en su zona de confort, es decir, si hay la temperatura óptima de aire y humedad. Este sensor es de importancia vital, puede ayudar a personas con problemas respiratorios volviéndolo uno de los más completos en el mundo de los sensores (ibíd.).

1.3.9. Sensor de ritmo cardiaco o Pulsómetro

Como su propio nombre indica, este sensor permite conocer las pulsaciones del corazón. Además, según el terminal, brinda información sobre el estrés que sufre en ese momento el cuerpo y, por ejemplo, el nivel de oxígeno en sangre.
El Galaxy S5, lo lleva incorporado en su parte trasera para determinar el ritmo cardiaco del usuario a través de los vasos sanguíneos de los dedos. El funcionamiento de este sensor se basa en los cambios de color cuando el dedo está iluminado por el flash LED. O sea, cuando se tiene una pulsación hay un momento donde el dedo oscurece y ese cambio, lo detecta la aplicación como aparece reflejado en la figura 1.6 (ibíd.).

Figura 1.6. Empleo del pulsómetro

1.3.10. Detector de huellas

Se trata de las últimas características incorporadas para temas de seguridad móvil. Los primeros teléfonos que comenzaron a incluir este sensor de huellas dactilares que permite darle muchos usos, como bloquearse, desbloquearse, permitir pagos electrónicos y mucho más.
Este sensor se encarga de escanear la huella dactilar propia y única de cada persona y la utiliza como método de desbloqueo. Ya no importaría olvidarse del patrón, PIN o contraseñas y se tendría una contraseña única, en este caso, la huella. Esta serie de detectores se basan en sensores capacitivos, que se encargan de reconocer cada línea de las huellas y las recopila en una imagen digital que almacenarán para compararla con cada dedo que intente desbloquear el teléfono móvil, a continuación, aparece un ejemplo en la figura 1.7 (Gallego, 2015).

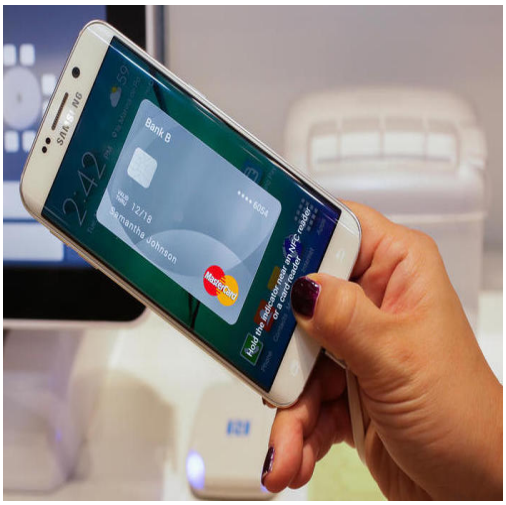


Figura 1.7. Detector de huellas

1.4. Plugins

Un plugin es aquel componente que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software. En español puede nombrarse al plugin como un complemento a pesar de que no forma parte del diccionario de la Real Academia Española pues se trata de un concepto que pertenece a la lengua inglesa. Normalmente un plugin se ejecuta por el programa principal a través de una interfaz.
Una de las principales ventajas que ofrecen estos complementos es que facilitan la colaboración de los que interactúan con él al trabajar sobre una estructura de bajo acoplamiento. En ocasiones pueden ocurrir conflictos entre un plugin y la aplicación principal, provocando diversos fallos. Mayormente cuando ocurre esto, el software brinda la opción de desactivar o desinstalar el plugin (María Merino, 2013).

1.5. Orientación Espacial

Es la habilidad natural que poseen todos para mantener la orientación del cuerpo y la postura en relación al espacio físico que les rodea. Esta capacidad permite, no solamente situar en el espacio, encontrar caminos o leer mapas, sino también crear los modelos mentales necesarios para desarrollar actividades en las que se tienen variables de dimensión y dirección. La evolución de la tecnología, propone dos reflexiones en direcciones contrarias. La primera consiste en la operación que ocurre cuando los sistemas mecánicos pasan por automatismos, con la inclusión de mecanismos electrónicos, hasta abandonar su condición de objetos físicos y algunas de sus partes se hacen virtuales. La segunda es la aparición de los sistemas de localización virtual en teléfonos móviles, a un cambio fundamental en el objeto de la localización. Aquella necesidad de orientación que llevó al uso de sistemas digitales muy próximos, ahora permite que uno sea el localizado (Polo, 2011).

La plataforma Android proporciona varios sensores que permiten a las aplicaciones monitorear el movimiento o la posición de un dispositivo. Los sensores de movimiento como el acelerómetro o el giroscopio son útiles para controlar el movimiento del dispositivo, como inclinación, movimiento, rotación o balanceo. Los sensores de posición son útiles para determinar la posición física de un dispositivo en relación con la Tierra. Un uso común de los sensores de movimiento y posición, especialmente para juegos, es determinar la orientación del dispositivo, es decir, el rumbo del dispositivo (norte / sur / este / oeste) e inclinarse. El acelerómetro mide las fuerzas de aceleración en el dispositivo; es decir, mide qué tan rápido se acelera el dispositivo y en qué dirección. El magnetómetro, mide la fuerza de los campos magnéticos alrededor del dispositivo, incluido el campo magnético de la Tierra. Puede usar el magnetómetro para encontrar la posición del dispositivo con respecto al mundo externo. Con la información obtenida a través del acelerómetro y el magnetómetro es posible determinar la orientación.

El acelerómetro y el giroscopio son dos sensores esenciales para el funcionamiento de los dispositivos móviles, a continuación, se muestran varias diferencias que estos poseen: El acelerómetro permite detectar la orientación del teléfono, y rotar la pantalla en función de esta; El giroscopio es el encargado de detectar los gestos y movimientos que se realizan con el célular para transmitirlo a la aplicación que se está utilizando. El acelerómetro es capaz de medir la orientación de un cuerpo con respecto a la superficie de la Tierra (el ángulo) pero de forma estacionaria; lo que no puede medir es la velocidad y el tiempo de permanencia en que se aplica; en un avión, en el momento del aterrizaje el acelerómetro sabe que se está bajando, pero no puede identificar por cuanto tiempo ni a qué velocidad. El Giroscopio puede hacerlo (Giroscopio, 2017).

1.5.1. Sistema de coordenadas de sensores en Android

Al utilizar los sensores de orientación y movimiento en Android, se definen dos sistemas de coordenadas: el sistema de coordenadas globales xE, yE, zE y un sistema de coordenadas del dispositivo x, y, z. Ambos sistemas de coordenadas se ilustran en la figura 1.8. Esta figura muestra el dispositivo colocado en el ecuador de la Tierra, con cierta inclinación con respecto a la Tierra. Todos los sistemas de coordenadas para sensores

de tres ejes obedecen estos sistemas de coordenadas, excepto Sensor.TYPE_ORIENTATION, que está en desuso (Milette y Stroud, 2010).

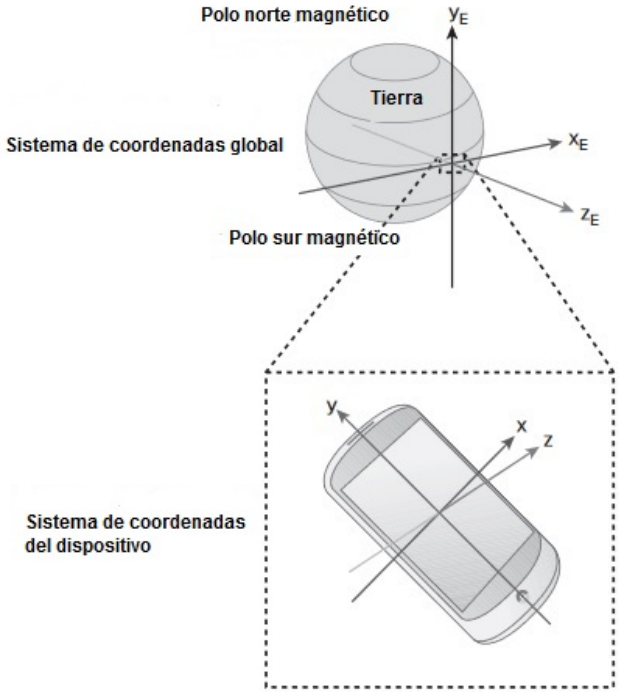


Figura 1.8. Sistema de coordenadas globales y en el dispositivo.

Sistema de coordenadas globales

Todos los sensores y métodos que se refieren a una orientación absoluta con respecto a la Tierra (excepto el sensor de orientación) utilizan el sistema de coordenadas global (ibíd.)]. Éstas incluyen:

- El sensor de vector de rotación, que utiliza el acelerómetro, el magnetómetro y el giroscopio para detectar la orientación del dispositivo en relación con la Tierra.
- *getRotationMatrix()*, *getRotationMatrixFromVector()* y *getQuaternionFromVector()*, que obtienen la matriz de rotación o el *quaternion* correspondiente a la rotación del dispositivo que puede asignar el sistema de coordenadas del dispositivo al sistema de coordenadas global.
- *getOrientation()*, que toma una matriz de rotación generada a partir de *getRotationMatrix()* y devuelve un vector de orientación.
- *getInclination()*, que toma una matriz de rotación generada a partir de *getRotationMatrix()* y devuelve la inclinación magnética.

En el sistema de coordenadas global las xE apuntan aproximadamente al este-paralelo a la superficie de la tierra, pero a 90 grados de yE; yE apunta hacia el norte magnético y zE apunta a los puntos del centro de la tierra.

Sistema de coordenadas en el dispositivo

Los sensores inerciales de tres ejes sin procesar (acelerómetro, magnetómetro y giroscopio) informan los valores correspondientes al sistema de coordenadas del dispositivo. El sistema de coordenadas de sensores con 3 ejes se define en relación a la pantalla del dispositivo en su orientación natural. Para un dispositivo móvil la orientación predeterminada es vertical. Cuando un teléfono se sostiene en su orientación natural, el eje X es horizontal y apunta a la derecha, el eje Y es vertical y apunta hacia arriba y el eje Z apunta fuera de la pantalla. Destacar que lo más importante sobre el sistema de coordenadas de sensores es que nunca cambia cuando el dispositivo se mueve o cambia de orientación (Wei, 2013). Es necesario explorar el sistema de coordenadas de Unity para conocer las diferencias que existen entre estos dos entornos de desarrollo.

1.5.2. Sistema de coordenadas en Unity

Unity contiene tres tipos de coordenadas fundamentales a tener presente al crear una Interfaz Gráfica de Usuario (GUI). Es importante conocer las diferencias que poseen y sobretodo, tener en cuenta que sistema de coordenadas está manejando un determinado objeto de interfaz de la escena. Los tres tipos de coordenadas son: *Screen Space*, *Viewport Space* y *World Space* (David, 2013).

- ***Screen Space***
  Como lo dice su nombre este sistema de coordenadas está basado en píxeles. Esto significa que solo es necesario especificar dos valores (x, y) que indican el número de píxeles a contar desde la esquina inferior izquierda, hasta la esquina superior derecha. Tener presente que este sistema solo tiene sentido en la pantalla donde se ve el juego y no dentro de la escena 3D como tal.
- ***Viewport Space***
  Cada dispositivo digital que pueda poseer un videojuego contiene una resolución diferente. La resolución es la cantidad de píxeles horizontal y vertical que posee el dispositivo. Las resoluciones de las pantallas de los dispositivos son la cantidad de píxeles que hay disponibles para mostrar el juego o la aplicación que se va a desarrollar. Una cámara en Unity puede especificar exactamente en qué porción de la pantalla pretende mostrar lo que renderiza de la escena 3D. La propiedad *Normalized Viewport Rect* del componente *Camera*, indica a que porción de la pantalla se desea mostrar lo que la cámara renderiza. Esta propiedad contiene cuatro valores (x, y, w, h) que son valores que están entre 0 y 1, y muestran el rectángulo de la pantalla donde la cámara renderiza. Los valores (x, y) representan la esquina inferior izquierda del rectángulo, por defecto $(x, y) = (0, 0)$, al ser unidades normalizadas, significa $0 * 100\,\%$ del ancho del dispositivo, $0 * 100\,\%$ del alto del dispositivo, es decir el pixel $(0, 0)$ independiente de la resolución. Por otro lado (w, h) representan el ancho y alto de este mismo rectángulo, donde $(w, h) = (1, 1)$ significa $1 * 100\,\%$ del ancho del dispositivo, $1 * 100\,\%$ del alto del dispositivo, es decir el pixel (ancho, alto), que es independiente del dispositivo.
- ***World Space***
  El sistema de coordenadas de este mundo 3D es precisamente "el sistema de coordenadas del mundo",

y los ejes de este sistema se referencian en la esquina superior derecha donde se puede observar un pequeño cubo con tres conos de colores apuntando en distintas direcciones (X es rojo, Y es verde y Z es azul) como aparece en la figura 1.9.

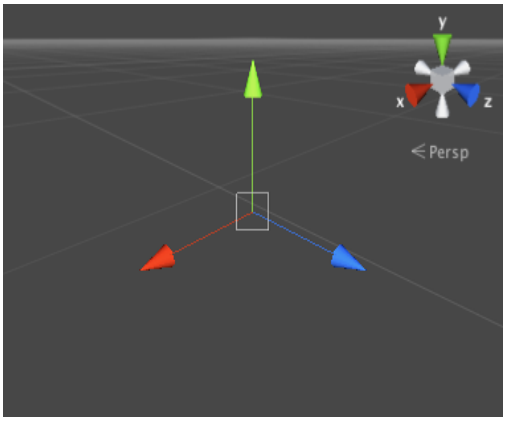


Figura 1.9. Sistema de coordenadas en Unity

Todos los *GameObjects* que aparecen en la escena tienen por defecto una componente *Transform*. No se puede tener un *GameObjects* sin un *Transform*. El *Transform* es usado para almacenar la posición, rotación y la escala (David, 2013). Unity para representar las rotaciones utiliza los *Quaternions*.

1.5.3. Quaternion

Los *quaternions* son una extensión de los números reales, similar a la de los números complejos. Mientras que los números complejos son una extensión de los reales por la adición de la unidad imaginaria i, tal que $i^2 = -1$, los *quaternions* son una extensión generada de manera análoga añadiendo las unidades imaginarias: i, j y k a los números reales, tal que $i^2 = j^2 = k^2 = -1$ (Favieri, 2008).

Definición de *quaternion*

Quaternion es un número de la forma $q = a + bi + cj + dk$ donde $a, b, c, d \in \Re$.
"a" se denomina parte real o escalar, y "$bi + cj + dk$" se denomina parte imaginaria o vectorial.

Valor absoluto o norma de un *quaternion*

Dado el *quaternion* $q = a + bi + cj + dk$ se define su norma o valor absoluto como $\left|q\right| = \sqrt{a^2 + b^2 + c^2 + d^2}$ (ibíd.) .

Normalización de un *quaternion*

Dado un *quaternion* cuya norma no sea igual a uno podemos normalizarlo definiendo un nuevo *quaternion*, asociado al primero, mediante la siguiente operación:

$$q_1 = \frac{q}{|q|}$$

Aplicaciones de los *quaternions*

Los *quaternions* no solo son una curiosidad algebraica. También tienen varias aplicaciones que van desde la teoría de los números, donde se pueden utilizar para probar resultados, como el teorema de Lagrange que dice que todo número natural n puede ser expresado como la suma de cuatro cuadrados perfectos, teoría de la relatividad, entre otras.

Los *quaternions* en física representan rotaciones en el espacio. También tienen aplicaciones en el electromagnetismo y la mecánica cuántica. Los *quaternions* se utilizan a menudo en gráficos por computadoras para representar la orientación de un objeto en un espacio tridimensional (Mendoza, 2010). Además, los *quaternions* evitan un fenómeno llamado *gimbal lock* que puede producirse cuando, por ejemplo, en sistemas de rotación de pitch/yaw/roll, la inclinación se gira 90 grados hacia arriba o hacia abajo, de modo que el pitch y el yaw corresponden al mismo movimiento, y el grado de la libertad de rotación se pierde.

1.5.4. Matriz de rotación

En álgebra lineal, una matriz de rotación es la matriz que representa una rotación en el espacio euclídeo. Como se muestra en la siguiente matriz (Rose, 2014):

$$R(\theta) = \begin{bmatrix} cos\theta & -sen\theta \\ sen\theta & cos\theta \end{bmatrix}$$

Representa la rotación de θ grados del plano en sentido antihorario. En tres dimensiones, las matrices de rotación representan las rotaciones de manera concisa y se usan frecuentemente en geometría, física e informática.

Una matriz de rotación rota un objeto sobre uno de los tres ejes de coordenadas, o cualquier vector arbitrario. Las siguientes matrices de rotación(3x3) realizan rotaciones de vectores alrededor de los ejes X, Y, o Z, se representan de la siguiente manera para expresar transformaciones que rotan puntos a través del ángulo θ en radianes sobre el origen de coordenadas (Goebel, 2017).

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sen\theta \\ 0 & sen\theta & cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sen\theta \\ 0 & 1 & 0 \\ -sen\theta & 0 & cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sen\theta & 0 \\ sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La matriz de rotación (4x4) es más compleja, ya que se necesita toda la matriz de (3x3) para expresar rotaciones complejas. La esquina inferior derecha de la matriz (4x4) es siempre 1:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sen\theta & 0 \\ 0 & sen\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Estas rotaciones básicas se realizan en sentido antihorario alrededor del eje X y considerando un sistema de coordenadas con la regla de la mano derecha.

1.6. Trabajos relacionados

En la actualidad los sensores han alcanzado un gran auge debido a su uso en las diferentes ramas de la ciencia y la tecnología, como la realidad virtual y la realidad aumentada, algunos proyectos importantes que explotan el uso de los sensores de orientación son:

- **Full tilt**
  Es una biblioteca de JavaScript basada en *Promises* que tiene compatibilidad con la orientación del dispositivo y los datos del sensor de movimiento. Full Tilt proporciona a los desarrolladores tres representaciones de la información brindada por los sensores de orientación de los dispositivos: *Quaternions* ajustados en pantalla, Matriz de rotación y ángulos de Euler, que se pueden usar para crear experiencias 2D o 3D en navegadores web. Estas aplicaciones funcionan consistentemente en los navegadores web de los móviles, porque son capaces de adquirir información de los sensores brindada por el sistema operativo. (Marttila, 2015). Destacar que para hacer posible la realidad aumentada, tanto en Vuforia como Google VR, Wikitude y Full Tilt necesitan utilizar los sensores del dispositivo para obtener la orientación, y poder colocar correctamente los elementos virtuales con respecto a la orientación real del dispositivo.
- **Vuforia**
  Es una Kit de Desarrollo de Software (SDK, por sus siglas en inglés) que permite construir aplicaciones basadas en la Realidad Aumentada; una aplicación desarrollada con Vuforia utiliza la pantalla del dispositivo como un "lente mágico" en donde se entrelazan elementos del mundo real con elementos virtuales. La cámara muestra a través de la pantalla del dispositivo, vistas del mundo real, combinados con objetos virtuales como: modelos, bloque de textos, imágenes, entre otros (Cruz, 2014). Una

aplicación desarrollada con este SDK puede brindar un reconocimiento de imágenes y textos como aparece en la figura 1.10.

Figura 1.10. Aplicación desarrollada con Vuforia

- **Wikitude**
  Wikitude es una aplicación gratuita de realidad aumentada disponible para dispositivos con sistema operativo BlackBerry, iPhone, Windows y Android. Con ella, gracias a las imágenes tomadas directamente desde los dispositivos móviles, se puede añadir una capa de información donde se podrán observar restaurantes, cafeterías o incluso, seguidores de Twitter que se encuentran próximos a la posición. Tan sencillo como usar la cámara del teléfono para explorar el entorno.
  Wikitude funciona mediante la cámara del móvil, el receptor de GPS y la brújula del móvil. Estas herramientas pueden saber la ubicación (GPS), hacia dónde se está orientado (brújula) y qué se está viendo (cámara), en la figura 1.11 se muestra un ejemplo (Aroca, 2012).

Figura 1.11. Muestra de cómo funciona Wikitude

- **Google-VR**
  Google ofrece a los desarrolladores dos plataformas de Realidad Virtual (VR): Cardboard, la plataforma de realidad virtual móvil más popular y accesible del mundo, y Daydream, una nueva plataforma para VR móvil de baja latencia, inmersiva e interactiva. Los SDK de Google-VR incluyen todo lo necesario para desarrollar estas aplicaciones, incluidas bibliotecas, documentación de APIs, ejemplos de desarrollador y directrices de diseño, en la figura 1.12, se puede visualizar un ejemplo del empleo

de Google-VR.

Figura 1.12. Google-VR

1.7. Marco de trabajo

Para llevar a cabo el desarrollo de un software se necesita analizar las herramientas que se van a emplear, el lenguaje de programación a utilizar, las plataformas y la metodología de desarrollo de software.

1.7.1. Lenguajes de programación.

- **Java**
  Java es tanto un lenguaje de programación, así como una plataforma informática, y un amplio abanico de tecnologías. Originalmente perteneció a la empresa Sun Microsystems y posteriormente fue comprada por la compañía Oracle. Es un lenguaje orientado a objetos, potente, versátil y multiplataforma. Java 8 fue elegido como el lenguaje para el entorno de desarrollo de Android para el presente trabajo. (Alvarez, 2001).
- **C#**
  El lenguaje de programación C# fue creado por el danés Anders Hejlsberg que diseñó también los lenguajes Turbo Pascal y Delphi. El C# es un lenguaje de programación orientado a objetos. Su código se puede tratar íntegramente como un objeto. Su sintaxis es muy similar a la de JAVA. Es un lenguaje orientado a objetos y a componentes. Ahorra tiempo en la programación ya que tiene una biblioteca de clases muy completa y bien diseñada(computacional, 2005). C# se utiliza como lenguaje de programación para el desarrollo de las aplicaciones de Unity que utilizan la biblioteca desarrollada en Android.

1.7.2. IDEs y Plataformas.

- **Android Studio**
  Android Studio es el Entorno de desarrollo Integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en IntelliJ IDEA (Studio, 2018). La versión de Android Studio utilizada es la 3.0, se escogió para desarrollar la biblioteca que implementa el acceso a los sensores, debido a las funciones que ofrece, las cuales aumentan la productividad durante la compilación de aplicaciones para Android, como las siguientes:
  - Un sistema de compilación basado en Gradle flexible.
  - Un emulador rápido con varias funciones.
  - Un entorno unificado en el que se puede realizar desarrollos para todos los dispositivos Android.
  - Instant Run para aplicar cambios mientras la aplicación se ejecuta sin la necesidad de compilar un nuevo Aplicación Empaquetada de Android (APK).
  - Integración de plantillas de código y GitHub para ayudar al usuario a compilar funciones comunes de las aplicaciones e importar ejemplos de código.
  - Gran cantidad de herramientas y *frameworks* de prueba.
- **Unity**
  Unity es un motor de videojuegos de nivel profesional, con un entorno de desarrollo integral, con capacidad de desarrollar juegos 2D y 3D, compatible con programas como 3dMax o Adobe Photoshop con el cual se pueden desarrollar videojuegos para casi cualquier plataforma, como Windows, Mac, Android y iPhone. Ofrece flexibilidad en cuanto al lenguaje de programación a utilizar, se pueden elegir entre C#, JavaScript o Boo (Yeeply, 2014). La versión de Unity utilizada para este proyecto es la 5.6.

1.7.3. Otras herramientas

- **Visual Paradigm**
  Visual Paradigm es una herramienta de Ingeniería de Software Asistida por Computadora (CASE, por sus siglas en inglés). Ella brinda un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Visual Paradigm ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque Orientado a Objetos (Sierra, 2007). La versión de Visual Paradigm utilizada fue la 8.0, se escogió para crear y diseñar los diagramas del trabajo, un ejemplo es el diagrama de componente creado para representar la arquitectura seleccionada.

- **Git**
  Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente, es decir Git proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida (Edgar, 2016). Uno de los fundamentos principales de Git es que se va a realizar un desarrollo de programación lineal, cada uno de los desarrolladores se va a encargar de programar su parte de funcionalidad que será almacenada en su disco duro local. En este trabajo se utilizó Git para llevar el control de los cambios realizados en Android Studio.

1.7.4. Metodología de desarrollo XP

La metodología XP fue desarrollada por Kent Beck, es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios (Joskowicz, 2008).
El objetivo de XP es tener grupos pequeños o medianos. Cuando los requerimientos aún son muy ambiguos y/o pueden ser de alto riesgo, se cambian durante el proceso. Esta metodología pretende tener al cliente vinculado directamente con el desarrollo del proyecto. También, se incrementa la comunicación entre el cliente y el desarrollador en el entorno de trabajo, con el motivo de mantener una agenda dinámica (Letelier, 2006).

1.8. Conclusiones parciales

En el presente capítulo se analizaron distintos sensores que contienen los dispositivos móviles y la función de cada uno de ellos, identificando al acelerómetro y el magnetómetro como la solución más viable para obtener una orientación correcta de los dispositivos. El análisis de conceptos como *quaternion* y matriz de rotación permitieron seleccionar estas variantes para manipular la rotación de los dispositivos, debido a que son las ideales, pues los *quaternions* evitan un fenómeno llamado *gimbal lock* y en Unity los *quaternions* se utilizan para representar las rotaciones. El estudio de los sistemas de coordenadas utilizados en Unity 3D y Android Studio permitió llegar a la conclusión de que es necesario convertir o traducir la rotación adquirida de un sistema a otro, para que tenga sentido dentro del uso de aplicaciones de realidad virtual o realidad aumentada.

CAPÍTULO 2

Características y diseño del sistema

2.1. Introducción al capítulo

En este capítulo se da respuesta al objetivo general del trabajo, proponiendo una serie de procedimientos para la creación de una biblioteca que contenga los sensores disponibles en Android. Luego, se diseña un plugin para el acceso a estas funcionalidades en Unity. También se implementa un método de corrección de la información del sensor de orientación. Se explica la ingeniería del sistema, donde se describen las historias de usuarios, el plan de estimación y las tarjetas Clase-Responsabilidad-Colaborador (CRC). Por otra parte, se explica brevemente la arquitectura basada en plugin de la solución.

2.2. Solución técnica

Para llevar a cabo el presente proyecto se estudiaron varios sensores: acelerómetro, sensor de campo magnético, sensor de luz, de proximidad, de temperatura ambiental, de presión, de humedad relativa y de orientación. Se podrán ver sus limitaciones a la hora de trabajar con ellos como un sensor en sí o usarlo en una aplicación más compleja. Los valores de estos sensores son almacenados en una biblioteca desarrollada en Android Studio, donde también se implementan una serie de métodos para adquirir la información de orientación.

Android presenta como funcionalidad heredada un sensor para la orientación, que está obsoleto (Sensor.TYPE_ORIENTATION). Este sensor de orientación es virtual, combina datos de otros sensores para determinar el rumbo y la inclinación del dispositivo. Debido a problemas con la precisión del algoritmo, este tipo de sensor ya no se utiliza. La forma recomendada de determinar la orientación del dispositivo implica utilizar el acelerómetro (Sensor.TYPE_ACCELEROMETER) y el sensor de campo geomagnético (Sensor.TYPE_MAGNETIC_FIELD). En la clase *SensorManager* se utilizan los métodos *getDefaultSensor(int Type)* para obtener el sensor predeterminado para un tipo dado, *registerListener(SensorEventListener listener, Sensor sensor, int samplingPeriodUs)* para registrar un *SensorEventListener* y *unregisterListener(SensorEventListener listener)* para anular el registro de un oyente.

Para obtener la orientación correcta se utiliza el método *getRotationMatrix()*, al cual se le pasan lo valores obtenidos del sensor campo magnético y del acelerómetro, y luego esta matriz de rotación a través de cálculos se convierte a *quaternion*. Este *quaternion* es creado para utilizarlo en los scripts de Unity, se calculó para poder dar una orientación correcta, debido a que Android trabaja con mano derecha y Unity con mano izquierda, los ejes de coordenadas de ambas herramientas están invertidos. En Unity se accede al *quaternion* creado en Android y se le realizan transformaciones para que tenga el mismo sentido de mano izquierda y que apunte hacia la parte trasera del teléfono. También se crea un plugin para Unity, donde se tendrá acceso a la biblioteca anterior. Este plugin funciona como puente entre las diferentes funcionalidades implementadas en Java y Unity, brindando así, la posibilidad de tener acceso a los demás sensores de los dispositivos móviles.
En el entorno de prueba se utilizaron varios dispositivos móviles y emuladores:

- Samsung Grand Prime SM-G531M, con Android 5.1.1
- BLU STUDIO 5.0 C, con Android 4.4.2
- LG-K430T, con Android 6.0
- El emulador Nexus 5X.
- Samsung Galaxy S4, Android 4.4.2

En la figura 2.1 se muestra un mapa conceptual en el que se pueden apreciar los módulos y dependencias de la solución en general:

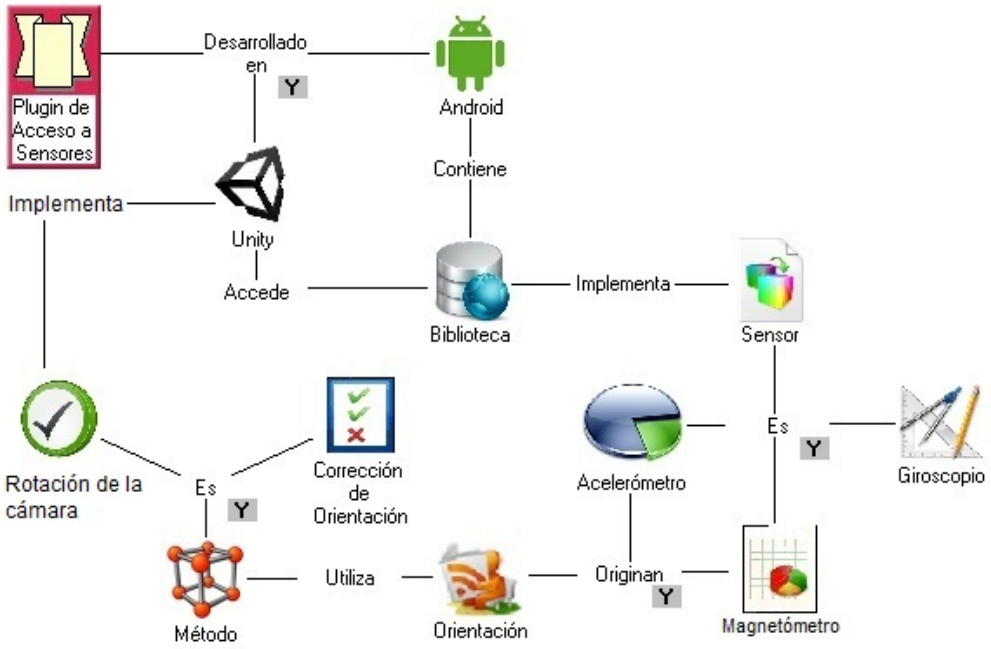


Figura 2.1. Mapa conceptual de la solución en general

2.2.1. Biblioteca Android

Para llevar a cabo el desarrollo de la biblioteca de Android se creó un nuevo proyecto en Android Studio que posee como nombre *AndroidSensor*. En el presente trabajo se creó una biblioteca a través de los módulos que contiene Android Studio, esta biblioteca es la encargada de llevar toda la información de los sensores en una clase java llamada *AndroidPlugin*. Los sensores utilizados por una aplicación deberán de estar declarados en el *AndroidManifest.xml* haciendo uso de la etiqueta *<uses-feature>* para que la aplicación pueda hacer uso de ellos. Está información es utilizada por el Android *Market* para saber si una aplicación es compatible con un dispositivo concreto. Se implementó también el patrón *observer*, con el cual no se sobrecarga el hilo principal de la aplicación, y la información de cada sensor se actualiza solamente cuando se lanza la notificación correspondiente a cada uno de ellos. Además, se implementó una aplicación extra, para Android, que utiliza esta biblioteca, de esta forma se comprueba su correcto funcionamiento sin dependencias de Unity. La aplicación muestra en la pantalla principal la información de todos los sensores existentes, y los diferentes métodos implementados para la adquisición de la información de orientación.
Para llevar a cabo el trabajo se utiliza la clase *SensorManager* de la SDK de Android. La tabla 2.1 incluye las clases e interfaces principales del *framework* para sensores (Wei, 2013).

Tabla 2.1. Framework para sensores de la plataforma Android.

Nombre	Tipo	Descripción
SensorManager	Clase	Se usa para crear una instancia del servicio de sensores. Proporciona varios métodos para el acceso a sensores, el registro y la eliminación de registros de las escuchas de eventos de sensores, entre otros.
Sensor	Clase	Se usa para crear la instancia de un sensor específico.
SensorEvent	Clase	El sistema lo usa para publicar datos del sensor. Incluye los valores de datos de sensores sin procesar, el tipo de sensor, la precisión de los datos y una marca de hora.
SensorEventListener	Interfaz	Esta interfaz proporciona métodos de llamada de regreso para recibir avisos del SensorManager cuando los datos o la precisión del sensor han cambiado.

2.2.2. Convertir una matriz de rotación a un *quaternion*

Los *quaternions* son una forma alternativa de representar la parte de rotación de una transformación, y pueden ser más fáciles de manipular que las matrices. Como con una matriz, se puede codificar transformaciones geométricas en una, concatenar varias de ellas para mezclar múltiples transformaciones, y aplicarlas a un vector, pero sólo pueden almacenar rotaciones puras. La gran ventaja es que puede interpolar precisamente entre dos *quaternions* para obtener una rotación parcial, evitando los enormes problemas de la interpolación más convencional con ángulos de Euler (Hargreaves, 2001). A continuación, se explican los pasos seguidos para transformar la matriz de rotación en *quaternions*.

Dada la matriz de rotación M

$$M = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

1. Primero que todo se calcula la traza de la matriz T (la suma de los elementos diagonales) a partir de la ecuación:
   $T = 4 - 4x^2 - 4y^2 - 4z^2$
   $= 4(1 - x^2 - y^2 - z^2)$
   $= x_{11} + x_{22} + x_{33} + 1$
2. Si la traza de la matriz es mayor que cero, entonces se debe realizar el siguiente cálculo:
   $S = 0.5/sqrt(T)$
   $W = 0.25/S$
   $X = (x_{32} - x_{23}) * S$
   $Y = (x_{13} - x_{31}) * S$
   $Z = (x_{21} - x_{12}) * S$
3. Si la traza de la matriz es menor o igual a cero, se identifica qué elemento de la diagonal principal tiene el mayor valor. Dependiendo de este valor, se calcula lo siguiente:
   $X = (x_{32} - x_{23}) * S$,
   $Y = (x_{13} - x_{31}) * S$,
   $Z = (x_{21} - x_{12}) * S$,
   $W = \frac{1}{(4 * S)}$;

   con $S = \frac{1}{2\sqrt{t+1}}$;

4. Si $(x_{11} > x_{22}) \wedge (x_{11} > x_{33})$:
   $X = \frac{1}{(4 * S)}$,
   $Y = (x_{21} + x_{12}) * S$,
   $Z = (x_{13} + x_{31}) * S$,
   $W = (x_{32} - x_{23}) * S$;

   con $S = \frac{1}{2\sqrt{1 + x_{11} - x_{22} - x_{33}}}$;

5. Si $x_{22} > x_{33}$:
   $X = (x_{21} + x_{12}) * S$,
   $Y = \frac{1}{(4 * S)}$,
   $Z = (x_{32} + x_{23}) * S$,
   $W = (x_{13} - x_{31}) * S$

$$\text{con } S = \frac{1}{2\sqrt{1 + x_{22} - x_{11} - x_{33}}};$$

6. En otro caso sería:
   $X = ((x_{13} + x_{31}) * S,$
   $Y = (x_{32} + x_{23}) * S,$
   $Z = \frac{1}{(4 * S)},$
   $W = (x_{21} - x_{12}) * S)$

   $$\text{con } S = \frac{1}{2\sqrt{1 + x_{33} - x_{11} - x_{22}}};$$

7. Luego de haber realizado la transformación de la matriz de rotación se obtiene el siguiente *quaternion*:
   $q = (x, y, z, w)$

Cambiar sentido de la rotación

Luego del análisis realizado y teniendo en cuenta las diferencias existentes en la orientación y basados en que el sistema de coordenadas de Android obtiene la rotación por la regla de la mano derecha y el sistema de coordenadas de Unity por la regla de la mano izquierda, para poder obtener un *quaternion* correcto se debe realizar una adaptación en la rotación, y se hace de la siguiente forma:
$q = (x, y, z, w) = (\hat{x}, \hat{y}, \hat{-z}, \hat{-w})$

En la figura 2.2 se puede visualizar el sistema de coordenadas de Unity y Android, según la regla que utiliza cada herramienta:

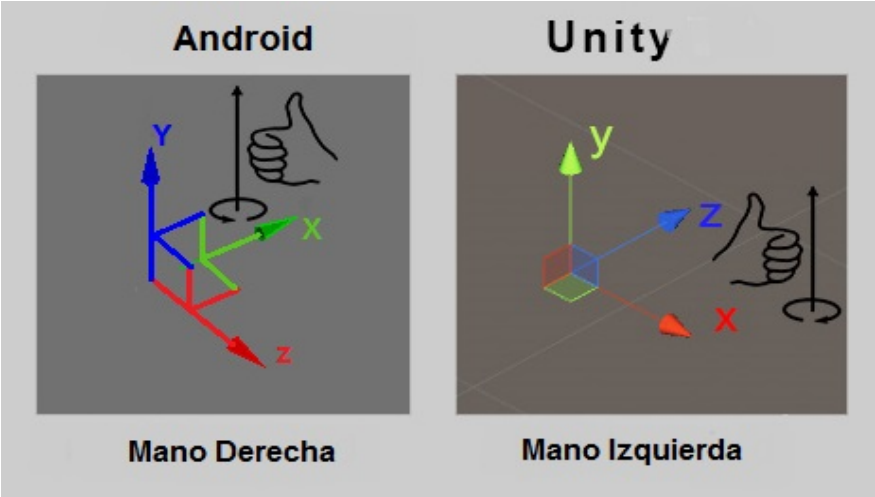


Figura 2.2. Sistema de coordenadas de Android y Unity

2.2.3. Plugin de Unity

Unity normalmente utiliza *scripts* en el proceso de desarrollo de una aplicación, pero también puede incluir código creado desde el exterior de Unity en forma de un plugin. Hay dos tipos de plugins que se pueden utilizar en Unity: *Managed plugins* y *Native plugins*.
Los *Managed plugins* son *assemblies* .NET creados con herramientas como Visual Studio o MonoDevelop. Estos contienen solo código .NET y no tienen acceso a características no soportadas por las bibliotecas .NET. Si el plugin es compilado afuera de Unity la fuente puede no estar disponible. Los *Native plugins* son bibliotecas de código nativo específico de ciertas plataformas. Estos pueden acceder a características como bibliotecas de código de terceros que por el contrario no están disponible en Unity (Unity, 2016).

Creando el Plugin

Para construir un plugin de Unity para Android, primero se debe crear una biblioteca compartida en Android Studio, una vez construida la biblioteca compartida se debe copiar a la carpeta ***Assets->Plugins->Android*** en el proyecto de Unity. Unity empacará los archivos .class junto con el resto del código Java y luego accede al código usando el Java Native Interface (JNI). JNI es utilizado tanto para llamar al código nativo desde Java, como para interactuar con Java desde el código nativo.

Utilización del plugin en Unity

Se crean dos *Scripts*, uno llamado *Sensor* para acceder a la biblioteca creada en Android Studio y otro *RotationController* para controlar la rotación de la cámara. En el *Script Sensor* se obtiene la información de la orientación del dispositivo brindada por la biblioteca Android. También, se implementa un método que traduce la rotación adquirida, la cual sigue la regla de la mano derecha, a una rotación que tenga sentido para Unity, que usa la regla de la mano izquierda. Es necesario aplicar esta transformación, pues, en caso de no hacerla, los movimientos de rotación del dispositivo se verían invertidos en Unity. Además, se implementa el script *RotationController*, el cual se asocia a la cámara de Unity, y le otorga la orientación real del dispositivo, esto crea el mismo efecto de las aplicaciones de Realidad Virtual y Realidad Aumentada mencionadas en el epígrafe Trabajos relacionados 1.6.

Vista de escena

Esta vista permite la visualización del aspecto gráfico en todo momento. Se implementa una escena en Unity como escenario de pruebas que contiene una cámara y diversos objetos, entre árboles y terreno. El escenario de pruebas consiste en utilizar la orientación adquirida por el plugin desarrollado en Android, y luego de las transformaciones pertinentes, asociarle en tiempo real la rotación a la cámara de Unity. De esta forma, se puede observar si el movimiento es correcto, pues la pantalla del teléfono, simula la pantalla de la cámara, y muestra los objetos de la escena que están dentro del rango de visión de la cámara. Además, la

cámara se puede mover hacia donde está enfocando, simulando así un paseo virtual como se muestra en la figura 2.3.

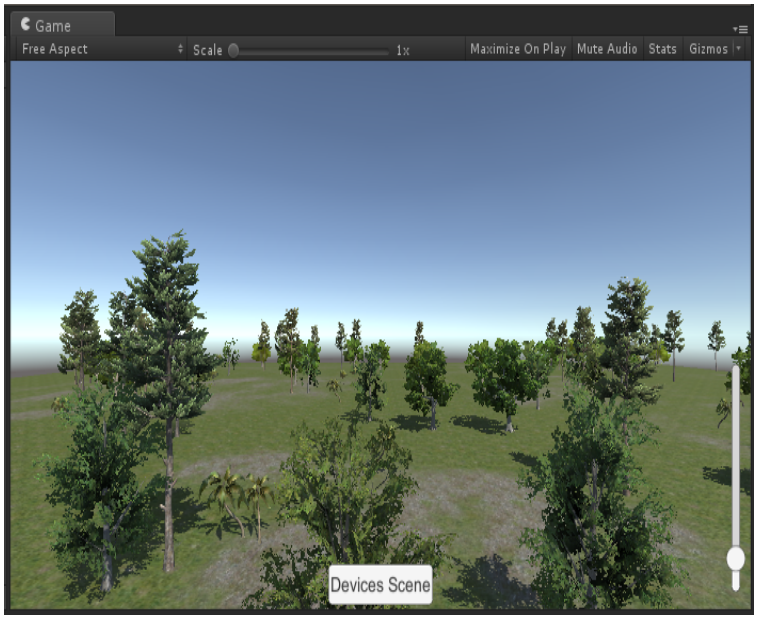


Figura 2.3. Vista de escena 1

Se implementa una segunda escena de Unity para visualizar los resultados de los métodos de orientación implementados en el plugin. En total se implementaron seis métodos, y cada uno de estos devuelve un quaternion asociado a la orientación del dispositivo:

- Giroscopio: depende del sensor giroscopio.
- *Quaternion from rotation matrix*: este método se utilizó para dar solución al problema existente con la orientación, se transformó la matriz de rotación en *quaternion*.
- *Modified orientation*: método de corrección manual, según la posición del teléfono (*portrait* o *landscape*)
- *Rotation vector*: método que describe un tipo de sensor sintético para calcular ángulo de rotación del sistema de coordenadas global con respecto al sistema de coordenadas del dispositivo utilizando el acelerómetro, el magnetómetro y el giroscopio si está disponible.
- *Game rotation vector*: método que es utilizado para describir un sensor, el cual refleja la rotación de un vector no calibrado.
- *Geomagnectic rotation*: este método se utiliza para describir un vector de rotación geomagnética.

Además, para comprobar qué tan cercanos son los datos obtenidos por los diferentes métodos, con respecto a la orientación brindada por el giroscopio (que es la más fiable, teniendo en cuenta lo explicado en el epígrafe giroscopio) 1.3.3, se haya el ángulo de diferencia entre ambas orientaciones, y se imprime en pantalla a forma de registro para ser utilizados en las pruebas del software, en la figura 2.4 se puede

visualizar como se conforma la escena.

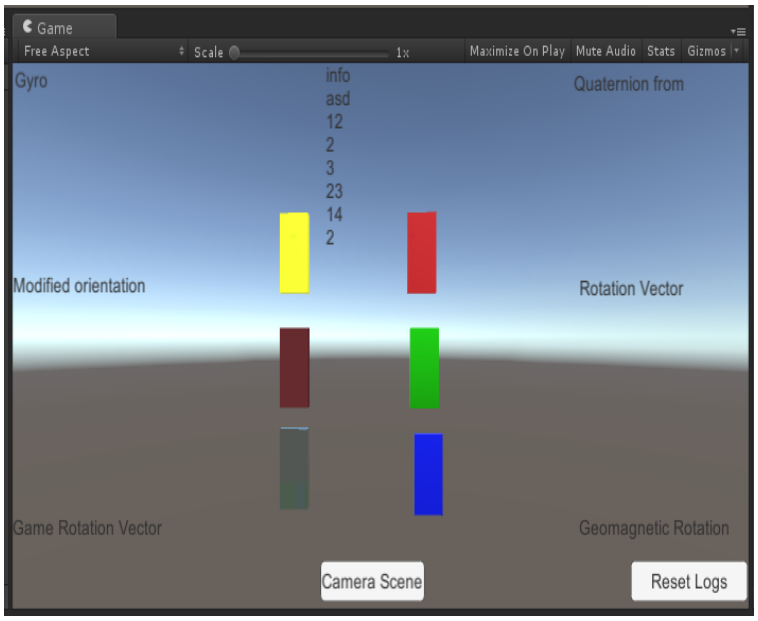


Figura 2.4. Vista de escena 2

2.3. Ingeniería del sistema

La metodología de desarrollo XP está compuesta por cuatro etapas, planificación, diseño, implementación y pruebas. En esta sección se mide el alcance general que tendrá el proyecto, a través de la redacción de las HU, el plan de estimación de esfuerzo por HU y el plan de duración de las iteraciones. Una vez realizado esto el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. La fase de exploración toma de algunas semanas a pocos meses, dependiendo del tamaño y familiarización de los programadores con la tecnología.

2.3.1. Historias de usuario

Una HU es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Estas se obtienen en la fase de diseño. Dentro de la metodología XP las HU deben ser escritas por los clientes, tal como ven ellos las necesidades del sistema. Las HU permiten responder rápidamente a los requisitos cambiantes.

- Número: número asignado a la HU.
- Nombre: nombre de la HU.
- Usuario: usuario del sistema que utiliza la HU.

- Prioridad en negocio: nivel de prioridad de la HU en el negocio. En caso de ser indispensable es alta, si no afecta el negocio en caso de no realizarse es media y baja si no constituye una prioridad.
- Riesgo en desarrollo: nivel de riesgo en caso de no realizarse la HU. Es alto cuando el riesgo implica en el funcionamiento de la plataforma. En caso de ser medianamente importante es medio el riesgo y bajo en caso de que no se considere un riesgo.
- Puntos estimados: estimación dada por el equipo de desarrollo del tiempo de duración de la HU, 1 equivale a una semana ideal de trabajo (5 días hábiles, 40 horas, 8 horas diarias).
- Iteración asignada: iteración a la que pertenece la HU.
- Programador responsable: responsable de la implementación de la HU.
- Descripción: describe lo que realizará la HU.
- Observaciones: casos especiales en el funcionamiento de la HU, flujos alternativos.

Tabla 2.2. Historia de usuario # 1

Historia de usuario	
Número: 1	**Nombre:** Mostrar lista de los sensores en un dispositivo móvil
Usuario: Especialista	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 2	**Iteración asignada:** 1
Programador responsable: Andro Rodríguez Martínez	
Descripción: El dispositivo debe mostrar una lista de los sensores que aparecen a continuación y los valores de estos en tiempo real: • Acelerómetro • Campo Magnético • Campo Magnético no calibrado • Luz Ambiental • Proximidad • Temperatura Ambiente • Presión Atmosférica • Humedad Relativa	
Observaciones: En caso de que el dispositivo no posea uno de los sensores mencionados no mostrará los valores	

Tabla 2.3. Historia de usuario # 2

Historia de usuario	
Número: 2	**Nombre:** Visualizar valores de la corrección de la Orientación
Usuario: Especialista	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta

Continúa en la próxima página

Tabla 2.3. Continuación de la página anterior

Puntos estimados: 2,8	**Iteración asignada:** 2
Programador responsable: Andro Rodríguez Martínez	
Descripción: El dispositivo debe mostrar los valores de los métodos que aparecen a continuación, estos fueron utilizados para obtener la orientación correcta del dispositivo: • Orientation Original • Orientation Corrected • Game Rotation • Geomagnetic Rotation • Orientation Rotation Indica la dirección a la que apunta el dispositivo, mide las velocidades de rotación.	
Observaciones: En caso de que el dispositivo no posea una de las funciones mencionadas no mostrará los valores	

Tabla 2.4. Historia de usuario # 3

Historia de usuario	
Número: 3	**Nombre:** Asociar rotación del dispositivo a cámara de Unity
Usuario: Usuario, Especialista	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 1,5	**Iteración asignada:** 3
Programador responsable: Andro Rodríguez Martínez	
Descripción: Se crea una escena con objetos (árboles), y se asocia la orientación del dispositivo a la rotación de la cámara. Así, cuando se mueva el dispositivo, se visualiza la escena hacia donde esté orientado.	
Observaciones: Puede existir ruido en el movimiento de la cámara, pues el sensor acelerómetro es muy sensible.	

Tabla 2.5. Historia de usuario # 4

Historia de usuario	
Número: 4	**Nombre:** Visualizar la rotación adquirida por los distintos métodos implementados
Usuario: Usuario, Especialista	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1,5	**Iteración asignada:** 3
Programador responsable: Andro Rodríguez Martínez	
Descripción: Se crea una escena en Unity con 6 rectángulos simulando la forma de dispositivos móviles. A cada uno de estos se asocia una orientación diferente, adquirida por los métodos implementados.	
Observaciones: Cada rectángulo funciona de forma individual al mover el dispositivo debido a que cada uno tiene implementado un método de orientación distinto haciendo uso de los sensores, algunos de estos rectángulos pueden permaneces estáticos, debido a que el teléfono no presenta determinado sensor, o el APIs de la versión de Android no lo soporta.	

2.3.2. Plan de Estimación de esfuerzo por historia de usuario

Teniendo en cuenta la prioridad que tiene una determinada HU en el desarrollo de la herramienta se decide en que iteración será implementada. Se define la estimación de las iteraciones que brinda cada HU, este plan tiene como objetivo mostrar la duración de las iteraciones que componen el proyecto.

Tabla 2.6. Estimación de esfuerzo por historia de usuario

Iteración	Historias de usuario		Puntos estimados (semanas)
1	1	Mostrar lista de los sensores en un dispositivo móvil	2
2	2	Visualizar valores de la corrección de la Orientación	2.8
3	3	Asociar rotación del dispositivo a cámara de Unity	1.5
	4	Visualizar la rotación adquirida por los distintos métodos implementados	1.5
Total			**7.8**

2.3.3. Plan de duración de las iteraciones

Una vez identificadas las HU y estimado el esfuerzo propuesto para el desarrollo del sistema de cada una de estas historias, se procede a la planificación de la etapa de implementación del sistema. El plan muestra la duración de la iteración y el orden en que serán implementadas las HU, que se define a través de sus dependencias con otras. En el proyecto se definieron tres iteraciones.

Tabla 2.7. Plan de duración de las iteraciones

Iteración	Historias de usuario		Duración (semanas)
1	1	Mostrar lista de los sensores en un dispositivo móvil	2
2	2	Visualizar valores de la corrección de la Orientación	2.8
3	3	Asociar rotación del dispositivo a cámara de Unity	3
	4	Visualizar la rotación adquirida por los distintos métodos implementados	
Total			**7.8**

2.4. Arquitectura y diseño

El análisis y diseño del sistema es un proceso fundamental y se ha realizado acorde a la metodología XP, esta no requiere la presentación del sistema a través de diagramas de clases utilizando la notación Lenguaje de Modelado Unificado (UML, por sus siglas en inglés), en su lugar se utilizan las tarjetas CRC. Los siguientes puntos muestran detalladamente como se ha realizado esta fase.

2.4.1. Arquitectura basada en plugins

Una arquitectura basada en plugins accede al código externo en ciertos puntos sin conocer todos los detalles de ese código. Además, puede ir desde lo más simple, como simplemente ejecutar archivos de comando en ciertos puntos, hasta lo más complejo. Los plugins deben seguir algún tipo de convención para funcionar. Esto puede ser como un formato de texto, un protocolo de red o una interfaz (Annand, 2015). Deben ser reconocibles y utilizables por el programa o paquete principal de alguna manera, ya sea que sigan una convención de la biblioteca y estén ubicados en un directorio particular o si alguien tiene que editar un archivo de configuración para señalarlos. Una arquitectura basada en plugins puede ser unidireccional, con el programa principal accediendo al plugin. En este caso, proporciona bibliotecas, interfaces y devoluciones de llamada. También se debe tener en cuenta que todo en un "plugin" no es necesariamente código. Los escritores de plugins a menudo envían recursos como imágenes, archivos de texto, entre otros. Con ese fin, a menudo se obtiene lo que se denomina SDK, que incluirá las APIs, los complementos y algún software para ayudar a crear, editar o importar recursos y reglas (ibíd.). A continuación, se muestra en la figura 2.5 un diagrama de componentes donde se refleja la arquitectura del sistema.

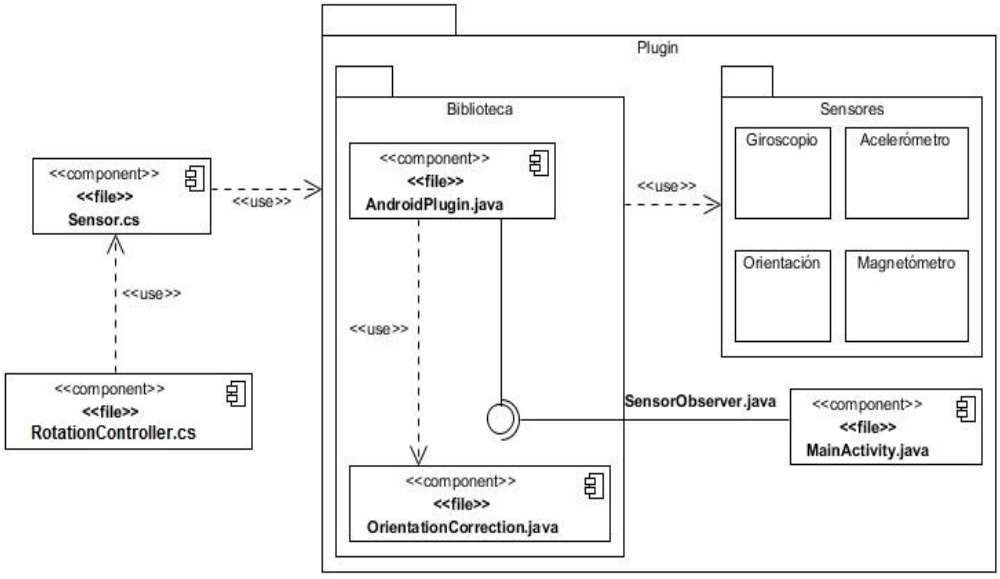


Figura 2.5. Diagrama de componentes

2.4.2. Tarjetas CRC

La manera de diseño y organización que se adopta para representar los módulos es diseñar una tarjeta CRC por cada uno de los módulos que brindan una funcionalidad directa al negocio. Las tarjetas CRC son un modelo simple de organizar las clases más importantes del sistema, representan las tareas de las HU. Estas presentan tres secciones:

- **Clase:** es cualquier persona, elemento, evento, concepto, pantalla o reporte. En fin, es el objeto que se analiza.
- **Responsabilidad:** función que la clase conoce o realiza, sus atributos y métodos.
- **Colaborador:** Conjunto de clases requeridas para que la clase reciba la información necesaria con el objetivo de completar una responsabilidad.

Tabla 2.8. Tarjeta CRC # 1

Tarjeta CRC	
Clase: AndroidPlugin	
Responsabilidad	**Colaboración**
• Esta clase es la encargada de llevar toda la información de los sensores. • *setContext(Context context)*: Este método obtiene una instancia del servicio *SensorManager*, luego usa el método *getDefaultSensor()* para recuperar sensores específicos y también registra cada sensor. • *onSensorChanged(SensorEvent event)*: Cada vez que el sensor realiza una medida, le envía los datos a este método en un objeto *SensorEvent*, que contiene los valores numéricos que se necesitan. • *notifyObservers(int sensorType)*: Se agrega una condición por cada sensor añadido y muestra esa información.	*MainActivity* *OrientationCorrection* *SensorObserver* *Sensor*

Tabla 2.9. Tarjeta CRC # 2

Tarjeta CRC	
Clase: MainActivity	
Responsabilidad	**Colaboración**

Continúa en la próxima página

Tabla 2.9. Continuación de la página anterior

• Muestra toda la información de los sensores en pantalla. • *onCreate(Bundle savedInstanceState)*: En este método se enlaza la interfaz de usuario con las variables llamando a la función *findViewById()*. Es el método que crea la actividad. • *onStart()*: Muestra al usuario la actividad y registra los oyentes. • *onPause()*: se ejecuta cuando una actividad va a dejar de estar en primer plano, para dar paso a otra.	*AndroidPlugin* *SensorObserver*

Tabla 2.10. Tarjeta CRC # 3

Tarjeta CRC	
Clase: OrientationCorrection	
Responsabilidad	**Colaboración**
• Se genera una matriz de rotación a partir del acelerómetro sin procesar y los datos del magnetómetro. • Agrega una nueva matriz de valores float para mantener la nueva matriz de rotación ajustada. • Se obtiene la rotación actual del dispositivo desde la pantalla y se agrega una declaración de cambio para ese valor. • *mDisplay.getRotation()*: Este método toma como argumentos la matriz de rotación original, los dos nuevos ejes en los que desea reasignar el eje X existente y el eje y, y una matriz para completar con los datos nuevos. • Se obtienen los ángulos de orientación de la matriz de rotación y se utiliza la nueva matriz. • Se crea un *quaternion* a través de la matriz de rotación	AndroidPlugin

Tabla 2.11. Tarjeta CRC # 4

Tarjeta CRC	
Clase: SensorObserver	
Responsabilidad	**Colaboración**
• En esta clase se añade una función de actualización por cada sensor.	*AndroidPlugin* *MainActivity*

Tabla 2.12. Tarjeta CRC # 5

Tarjeta CRC	
Clase: Sensor	
Responsabilidad	**Colaboración**
• Está clase funciona como puente de comunicación entre Android Studio y Unity, le brinda a Unity la posibilidad de acceder a la biblioteca creada en Android Studio, la cual contiene los sensores. • *start()*: Este método es llamado antes de cualquier actualización del primer cuadro de un objeto y solo es llamado una vez en la vida del *script*. • *Update()*: Este método es llamado antes de que el cuadro sea renderizado de forma automática por el sistema y antes de que las animaciones sean calculadas. • *getRotationVector()*: Devuelve la rotación del vector • *getQuaternionFromRotationMatrix()*: Retorna el *quaternion* obtenido de la matriz de rotación	*AndroidPlugin* *RotationController*

Tabla 2.13. Tarjeta CRC # 6

Tarjeta CRC	
Clase: RotationController	
Responsabilidad	**Colaboración**

Continúa en la próxima página

Tabla 2.13. Continuación de la página anterior

• En esta clase se obtiene un nuevo *quaternion* donde se invierte el eje Z y el W para que la rotación sea correcta, es la encargada de darle la orientación del dispositivo a la cámara. • Awake(): Este método se invoca en todos los objetos de la escena antes de llamar a la función de inicio de cualquier objeto. • quatDifference(Quaternion q1, Quaternion q2): Este método retorna la inversa del primer *quaternion* por el segundo. • GyroToUnity(Quaternion q): Este método devuelve un nuevo *quaternion*.	*Sensor*

2.4.3. Patrones de diseño

Un patrón es una unidad de información que expresa la relación entre tres componentes esenciales: un problema a resolver, una solución y un determinado contexto donde se repite una situación que vincula a los dos primeros componentes. Además, un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular (E.Gamma; Helm; Johnson y Vlissides, 1995). También destacar que un patrón de diseño es una solución a un problema en un contexto, que facilita la localización de los objetos que formarán el sistema, especifican las interfaces para las clases y facilita el aprendizaje y la comunicación entre programadores y diseñadores (Prieto, 2009). A continuación, se explica donde fueron utilizados los patrones de diseño:

Patrones GRASP

- **Experto:** Se refleja mediante cada una de las clases que componen el *AndroidSensorsTestApp*, debido a que cada clase contiene una información necesaria para realizar una labor específica.
- **Creador:** Este patrón se manifiesta a través de las clases *MainActivity* y *AndroidPlugin*. Estas clases tienen como responsabilidad crear instancias de las clases subordinadas, no en sentido contrario. La clase *MainActivity* utiliza a la clase *AndroidPlugin*, *AndroidPlugin* registra la interfaz *SensorObserver* y utiliza la clase *OrientationCorrection*.
- **Alta Cohesión y Bajo Acoplamiento:** Estos patrones tienen una estrecha relación entre sí, cada una de las responsabilidades de las clases están asignadas de tal manera, que cada una realice actividades específicas y exista la mínima dependencia con otras clases. Para reducir el acoplamiento se creó una interfaz *SensorObserver*, se ve reflejado también en la clase *OrientationCorrection* que tienen un

conjunto de funciones propias de ellas y el nivel de dependencia entre clases es más bajo.

Patrones GOF

- **Patrón Observer:** La utilidad de este patrón se refleja en los objetos observables y observadores. Los objetos observables implementan un mecanismo para registrar (*registerObserver*)y eliminar (*removeObserver*). Además, se declara un método (*notifyObservers*) que se encarga de avisar a todos los observadores de un cambio de estado. Los observadores son los objetos que se implementan en la interfaz *SensorObserver*, tendrá varios métodos (*void accelerometerUpdate(), void magneticUpdate()*, entre otros) que definen la reacción correspondiente al cambio de estado.
- **Patrón Singleton:** El uso de este patrón se refleja en la clase *AndroidPlugin* que tiene un constructor privado y un método llamado *getInstance()* que se encarga de crear una instancia de esta clase si aún no se ha creado. Asegura que solamente existirá una instancia de *AndroidPlugin*. El objetivo de este patrón es el de garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella.

2.5. Conclusiones parciales

En la elaboración de este capítulo se abordaron varios aspectos fundamentales del proyecto, donde se explica de forma breve como se desarrolló una biblioteca en Android Studio, la cual contiene acceso a varios sensores de Android a través del *SensorManager*. Los métodos utilizados para llevar a cabo la corrección del sensor de orientación, escogiendo como principales *getRotationMatrix()* y *getQuaternion()*, por ser los idóneos según lo explicado en el epígrafe Solución técnica 2.2. En Unity se utilizaron varias escenas para realizar las pruebas y así alcanzar un mejor resultado. Además, se vio de manifiesto la necesidad de implementar dos *scripts* en Unity, el *script Sensor* para acceder a la biblioteca creada en Android y *RotationController*, para controlar la rotación de una cámara con la propia orientación del dispositivo, y así crear un ambiente de realidad virtual que puede ser visualizado en dispositivos que no presentan el sensor giroscopio, siempre y cuando cuenten con el acelerómetro y magnetómetro.

CAPÍTULO 3

Implementación y pruebas

3.1. Introducción al capítulo

En el presente capítulo se aborda la elaboración y ejecución de los casos de pruebas realizados para validar los resultados alcanzados por el método de corrección de orientación y la implementación del sistema. Para el desarrollo de la fase de implementación se desglosan las HU en tareas de ingeniería, con el objetivo de encontrar los errores y corregirlos para obtener un producto final satisfactorio.

3.2. Implementación del sistema

Esta fase se lleva a cabo una vez concluida la fase de diseño, debido a que esta es la guía para comenzar a ejecutar los componentes y programar las funciones que deberá cumplir la aplicación (Montoya, 2012). A continuación, se describen las tareas de ingeniería correspondientes a las HU del sistema.

Tareas de ingeniería para la Iteración I

Para la primera iteración se definieron un total de dos tareas de ingeniería, las cuales le dan cumplimiento a la HU que muestra la lista de los valores de los sensores en un dispositivo móvil.

Tabla 3.1. Tarea de ingeniería # 1

Tarea	
Número de tarea: 1	**Número de Historia de usuario:** 1
Nombre de la tarea: Listar los valores de los sensores	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.6
Fecha de inicio: 19 de enero de 2018	**Fecha de fin:** 30 de enero de 2018
Programador responsable: Andro Rodríguez Martínez	
Descripción: Se crea la clase *AndroidPlugin* para obtener los valores en tiempo real de cada uno de los sensores implementados y que estos se actualicen constantemente.	

Tabla 3.2. Tarea de ingeniería # 2

Tarea	
Número de tarea: 2	**Número de Historia de usuario:** 1
Nombre de la tarea: Mostrar los valores de los sensores	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.4
Fecha de inicio: 2 de febrero de 2018	**Fecha de fin:** 5 de febrero de 2018
Programador responsable: Andro Rodríguez Martínez	
Descripción: Se crea la clase *MainActivity* para mostrar la información recopilada de cada uno de los sensores.	

Tareas de ingeniería para la Iteración II

En la segunda iteración se encuentra la tarea de ingeniería correspondiente a la HU que muestra los valores de la corrección de la orientación.

Tabla 3.3. Tarea de ingeniería # 3

Tarea	
Número de tarea: 3	**Número de Historia de usuario:** 2
Nombre de la tarea: Realizar la corrección de la orientación	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2.8
Fecha de inicio: 7 de febrero de 2018	**Fecha de fin:** 26 de febrero de 2018
Programador responsable: Andro Rodríguez Martínez	
Descripción: Esta clase se encarga de obtener la orientación en la que se encuentra el dispositivo, hallar una matriz de rotación y calcular un *quaternion* para traducir la información de la orientación.	

Tareas de ingeniería para la Iteración III

En la tercera iteración se definieron tres tareas de ingeniería, las cuales le dan cumplimiento a las HU asociadas a ellas.

Tabla 3.4. Tarea de ingeniería # 4

Tarea	
Número de tarea: 4	**Número de Historia de usuario:** 3
Nombre de la tarea: Mostrar la escena de giro de la cámara	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.0
Fecha de inicio: 28 de febrero de 2018	**Fecha de fin:** 7 de marzo de 2018
Programador responsable: Andro Rodríguez Martínez	

Continúa en la próxima página

Tabla 3.4. Continuación de la página anterior

Descripción: Esta vista representa una escena donde aparece un terreno con árboles. La vista debe mostrar el giro que realiza la cámara cuando se mueve el dispositivo móvil hacia cualquier dirección.

Tabla 3.5. Tarea de ingeniería # 5

Tarea	
Número de tarea: 5	**Número de Historia de usuario:** 3
Nombre de la tarea: Realizar giro de la cámara	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha de inicio: 8 de marzo de 2018	**Fecha de fin:** 12 de marzo de 2018
Programador responsable: Andro Rodríguez Martínez	
Descripción: Esta clase realiza el giro de la cámara según la orientación del dispositivo y cambia la dirección del *quaternion* calculado.	

Tabla 3.6. Tarea de ingeniería # 6

Tarea	
Número de tarea: 6	**Número de Historia de usuario:** 4
Nombre de la tarea: Mostrar la información de la diferencia entre los ángulos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.5
Fecha de inicio: 14 de marzo de 2018	**Fecha de fin:** 26 de marzo de 2018
Programador responsable: Andro Rodríguez Martínez	
Descripción: Se desarrolla una vista para mostrar la información obtenida de la diferencia entre los ángulos del *quaternion* del giroscopio y el *quaternion* de los métodos de corrección de orientación implementados.	

3.3. Pruebas de software

Las pruebas son un elemento crítico para la calidad del software. La importancia de los costos asociados a los errores, promueve la definición y aplicación de un proceso de pruebas minuciosas y bien planificadas. Las pruebas permiten validar y verificar el software, entendiendo como validación del software el proceso, externo al equipo de desarrollo, que determina si el software satisface los requisitos, y verificación como el proceso interno que determina si los productos de una fase satisfacen las condiciones de dicha fase (Pressman, 2002).

3.3.1. Pruebas de aceptación

Estas pruebas derivan de las HU que se han implementado como parte de la liberación del software. Una HU puede tener varias pruebas de aceptación para asegurar su funcionamiento (Pressman, 2002).
La prueba de aceptación es ejecutada antes de que la aplicación sea instalada dentro de un ambiente de producción. El cliente basado en esta prueba determina si acepta o rechaza el sistema. Las pruebas de aceptación le sirven al usuario para poder validar si el producto final se ajusta a los requisitos fijados, es decir, si el producto está listo para ser implantado para el uso operativo en el entorno del usuario.

Pruebas de aceptación para la Iteración I

Tabla 3.7. Prueba de aceptación # 1

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de usuario:** 1
Nombre: Listar los sensores en un dispositivo móvil.	
Descripción: Lista de los sensores que posee el dispositivo.	
Condiciones de ejecución: • El usuario deberá tener un dispositivo móvil con sistema operativo Android.	
Pasos de ejecución: 1. El usuario desde el dispositivo móvil debe abrir la aplicación *AndroidSensors*.	
Resultados esperados: Se muestra un listado con los sensores que contiene el dispositivo.	

Tabla 3.8. Prueba de aceptación # 2

Caso de prueba de aceptación	
Código: HU1_P2	**Historia de usuario:** 1
Nombre: Mostrar valores de los sensores en un dispositivo móvil.	
Descripción: Mostrar valores de los sensores que posee el dispositivo móvil.	
Condiciones de ejecución: • El usuario deberá tener un dispositivo móvil con sistema operativo Android.	
Pasos de ejecución: 1. El usuario desde el dispositivo móvil debe abrir la aplicación *AndroidSensors*. 2. El usuario debe mover el dispositivo para visualizar como se realizan cambios en los valores de los sensores.	
Resultados esperados: Se muestra un listado con los sensores que contiene el dispositivo y sus valores en tiempo real.	

Pruebas de aceptación para la Iteración II

Tabla 3.9. Prueba de aceptación # 3

Caso de prueba de aceptación	
Código: HU2_P1	**Historia de usuario:** 2
Nombre: Visualizar la orientación correcta del dispositivo móvil.	
Descripción: Prueba de la funcionalidad de la corrección de la orientación.	
Condiciones de ejecución: • El usuario deberá poseer un dispositivo móvil con sistema operativo Android. • El dispositivo deberá tener el sensor acelerómetro y el sensor magnetómetro.	
Pasos de ejecución: 1. El usuario desde el dispositivo móvil debe abrir la aplicación *AndroidSensors*. 2. El usuario debe mover el dispositivo en varias direcciones para observar cómo se realizan cambios en los valores de la orientación.	
Resultados esperados: Se muestra los tres ejes (azimut, pitch y roll) que contiene el dispositivo y sus valores según la rotación que se realice.	

Pruebas de aceptación para la Iteración III

Tabla 3.10. Prueba de aceptación # 4

Caso de prueba de aceptación	
Código: HU3_P1	**Historia de usuario:** 3
Nombre: Mostrar escena de un plano con objetos (árboles).	
Descripción: Prueba de la funcionalidad de la rotación correcta de la cámara.	
Condiciones de ejecución: • El usuario deberá poseer un dispositivo móvil con sistema operativo Android. • El dispositivo deberá tener el sensor acelerómetro y el sensor magnetómetro.	
Pasos de ejecución: 1. El usuario desde el dispositivo móvil debe abrir la aplicación *AndroidOrientation*. 2. El usuario para desplazarse a través de la escena debe tocar la pantalla del dispositivo. 3. El usuario da clic al botón *Devices Scene* y se direcciona a una vista con otra escena.	
Resultados esperados: Se muestra una escena con objetos (árboles) y un botón. También accede a una segunda escena a través del botón *Devices Scene*.	

Tabla 3.11. Prueba de aceptación # 5

Caso de prueba de aceptación	
Código: HU3_P2	**Historia de usuario:** 3
Nombre: Visualizar la rotación de la cámara.	
Descripción: Prueba de la funcionalidad de la rotación correcta de la cámara.	
Condiciones de ejecución: • El usuario deberá poseer un dispositivo móvil con sistema operativo Android. • El dispositivo deberá tener el sensor acelerómetro y el sensor magnetómetro.	
Pasos de ejecución: 1. El usuario desde el dispositivo móvil debe abrir la aplicación *AndroidOrientation*. 2. El usuario para hacer rotar la cámara de la aplicación debe mover el dispositivo hacia cualquier dirección.	
Resultados esperados: Se muestra como rota la cámara hacia la dirección que orienta el usuario.	

Tabla 3.12. Prueba de aceptación # 6

Caso de prueba de aceptación	
Código: HU4_P1	**Historia de usuario:** 4
Nombre: Mostrar escena con seis rectángulos, cada uno con una función distinta	
Descripción: Prueba para la funcionalidad: obtener valores de la diferencia angular entre el giroscopio y el método de corrección de la orientación.	
Condiciones de ejecución: • El usuario deberá poseer un dispositivo móvil con sistema operativo Android. • El dispositivo deberá tener el sensor acelerómetro y el sensor magnetómetro. • El dispositivo deberá tener el sensor giroscopio.	
Pasos de ejecución: 1. El usuario entra a la vista de la escena que muestra dos botones y seis rectángulos simulando la forma de dispositivos móviles. 2. El usuario debe girar el dispositivo para comprobar si los rectángulos giran en la misma dirección. 3. El usuario da clic en el botón *Camera Scene* para volver a la escena anterior.	
Resultados esperados: Se muestra una escena con dos botones y seis rectángulos simulando la forma de dispositivos móviles. Además, retorna a la primera escena mediante el botón *Camera Scene*.	

Tabla 3.13. Prueba de aceptación # 7

Caso de prueba de aceptación	
Código: HU4_P2	**Historia de usuario:** 4
Nombre: Mostrar valores de la diferencia angular entre el sensor giroscopio y el método de corrección de la orientación implementado.	
Descripción: Prueba para la funcionalidad: obtener valores de la diferencia angular entre el giroscopio y el método de corrección de la orientación.	
Condiciones de ejecución: • El usuario deberá poseer un dispositivo móvil con sistema operativo Android. • El dispositivo deberá tener el sensor acelerómetro y el sensor magnetómetro. • El dispositivo deberá tener el sensor giroscopio.	
Pasos de ejecución: 1. El usuario entra a la vista de la escena que muestra seis rectángulos simulando la forma de dispositivos móviles. 2. El usuario debe girar el dispositivo en los ejes X, Y y Z para obtener los valores de la diferencia angular correspondiente a cada eje. 3. El usuario para obtener nuevos valores de la diferencia angular debe dar clic en el botón *Reset Logs*.	
Resultados esperados: Se muestra de manera automática los valores de la diferencia angular entre el giroscopio y el método implementado. Además, reinicia los valores de la diferencia angular a través del botón *Reset Logs*.	

3.3.2. Resultados de las pruebas de aceptación

Durante el proceso de pruebas, se desarrollaron siete casos de pruebas de aceptación. Las pruebas de aceptación se realizaron de manera organizada por cada una de las iteraciones definidas. En la primera iteración se realizaron dos casos de pruebas, se detectaron inconformidades por parte de los usuarios, debido a que varios de los sensores desarrollados mostraban valores iguales. Esto posibilitó corregir el error y cambiar las variables de cada sensor para así poder continuar con la segunda iteración. Para la segunda iteración se realizó solo un caso de prueba, en el cuál se encontraron tres no conformidades, se detectó que el método de corrección de la orientación implementado en Android no era compatible con la rotación en Unity, debido a que los sistemas de coordenadas eran diferentes. Lo que permitió realizar varias transformaciones para corregir el error, abriendo paso así a la elaboración de la tercera iteración. En la tercera iteración se desarrollaron un total de cuatro pruebas, los resultados obtenidos no mostraron inconformidades. La aplicación de estas pruebas permitió comprobar la correcta implementación de las HU definidas con anterioridad. En la figura 3.1 se muestra una gráfica en la que se puede visualizar los resultados obtenidos de las pruebas de aceptación.

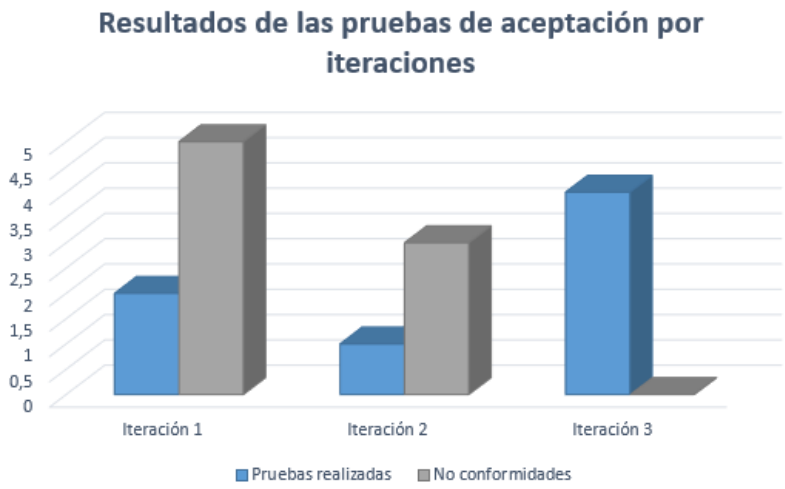


Figura 3.1. Resultados de las pruebas de aceptación

3.3.3. Pruebas al método de corrección de la orientación

Para obtener una medida precisa del método de corrección de la información del sensor de orientación implementado, se halló la diferencia angular con respecto al sensor giroscopio, la información obtenida para ambos pertenecen a la misma orientación, pues se adquieren simultáneamente desde el mismo dispositivo. Para hallar la diferencia angular se utilizó como dispositivo de prueba un Samsung Galaxy S4, con versión de Android 4.4.2, se obtuvieron un total de 210 valores, a través de la rotación realizada en el sistema de coordenadas X, Y y Z.

Los resultados representados en la tabla 3.14 muestran que la media de la diferencia angular entre el sensor giroscopio y el método implementado fue de $7,282$ grados en el eje X, $8,836$ grados para el eje Y y $5,069$ grados para el eje Z. La mediana obtenida por cada eje, está por debajo de la media, esto significa que, en la mayoría de los casos, la diferencia angular existente es pequeña, pero existen casos aislados donde la diferencia angular es considerable, según la orientación que posea el dispositivo de prueba. Los valores máximos y mínimos obtenidos muestran que la diferencia angular puede variar considerablemente en los tres ejes de coordenadas.

Tabla 3.14. Análisis de la diferencia angular entre el giroscopio y el método *OrientationCorrection*.

Ejes	Media	Mediana	Máximo	Mínimo
X	7,282	6,117	22,235	0,508
Y	8,836	8,431	28,485	0,023
Z	5,069	3,948	13,182	0,428

3.4. Conclusiones parciales

En el desarrollo de este capítulo se trabajó en la fase de implementación del sistema y en las pruebas de software durante la elaboración del mismo. Se obtuvieron resultados considerables de la diferencia angular entre el método de corrección implementado y el sensor giroscopio, teniendo en cuenta que el método implementado para la solución, utiliza el sensor acelerómetro y dicho sensor posee “ruido”. Estos resultados se pueden mejorar si se le elimina el “ruido” al acelerómetro.

Conclusiones

Con el desarrollo de la presente investigación se obtuvo un plugin en Unity para acceder a los sensores de Android, para lo cual:

- Se demostró q es posible obtener el acceso a todos los sensores de Android desde Unity mediante un Plugin, permitiendo a dispositivos móviles de gama media y baja, adquirir una orientación del dispositivo bastante fiable sin tener el sensor giroscopio.
- Los métodos más fiables para obtener la orientación del dispositivo es el *getRotationMatrix*, que elimina el problema del *gimbal lock* y el *getQuaternion*, que transforma la matriz de rotación en *quaternion*. El método implementado se comparó con el sensor giroscopio, logrando como resultado una diferencia angular considerable en los ejes de coordenadas X, Y y Z debido a que el sensor acelerómetro contiene "ruido".
- Se demostró la necesidad de implementar dos *scripts* en Unity, el *script Sensor* para acceder a la biblioteca creada en Android y *RotationController*, para controlar la rotación de una cámara con la propia orientación del dispositivo, y así crear un ambiente de realidad virtual que puede ser visualizado en dispositivos que no presentan el sensor giroscopio, siempre y cuando cuenten con el acelerómetro y magnetómetro.

Recomendaciones

La elaboración de este trabajo puede contribuir a futuras investigaciones en el área de realidad aumentada. Algunas recomendaciones para futuros trabajos son:

- Se recomienda la implementación de un método para el suavizado del movimiento asociado a la orientación adquirida, pues al ser el sensor acelerómetro tan sensible, no es viable en el uso en aplicaciones de inmersión, pues podría provocar mareos o desorientación.
- Se recomienda agregar al presente trabajo funcionalidades de geolocalización, para construir aplicaciones más completas y atractivas, teniendo como perspectiva las herramientas estudiadas Vuforia, Wikitude, Google VR y Full Tilt que explotan todas las posibilidades de los dispositivos móviles.

Glosario

GameObjects son objetos fundamentales en Unity que representan personajes, accesorios, y el escenario. 15

gimbal lock es un efecto molesto que se produce cuando se rota cualquier vértice o entidad en el eje Y 90/ − 90 grados. La consecuencia de esto es que el eje X y el Z acaban apuntando en la misma dirección, por lo cual se pierde el control sobre ellos. 16, 21, 48

Linux es un sistema operativo de software libre. 1, 4, 5

microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. 6

patrón *observer* es un patrón de diseño de software que define una dependencia del tipo uno a muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes. 24

ángulos de Euler constituyen un conjunto de tres coordenadas angulares que sirven para especificar la orientación de un sistema de referencia de ejes ortogonales, normalmente móvil, respecto a otro sistema de referencia de ejes ortogonales normalmente fijos. 17, 24

Acrónimos

APIs Interfaz de Programación de Aplicaciones. 5, 18, 31, 33
APK Aplicación Empaquetada de Android. 20

CASE Ingeniería de Software Asistida por Computadora. 20
CRC Clase-Responsabilidad-Colaborador. 22, 32, 33

GUI Interfaz Gráfica de Usuario. 14

HU Historia de Usuarios. 3, 29, 30, 32, 33, 39, 40, 42, 45

IDE Entorno de desarrollo Integrado. 20

JNI Java Native Interface. 27

OHA Alianza Abierta de Teléfonos. 4

SDK Kit de Desarrollo de Software. 17, 18, 24, 33

UCI Universidad de las Ciencias Informáticas. 1
UML Lenguaje de Modelado Unificado. 32

VERTEX Centro de Entornos Interactivos 3D. 1
VR Realidad Virtual. 18

XP Programación Extrema. 21, 29, 32

Referencias bibliográficas

ALVAREZ, Miguel Angel, 2001. Que es Java (vid. pág. 19).

ÁLVARO BORREGO, Manuel Báez y, 2012. *Introducción a Android*. Ed. por GRUPO TECNOLOGÍA UCM, Victoria López y. E.M.E. Editorial (vid. pág. 4).

ANA GARDEY, Julián Pérez Porto y, 2010. *Definición de sensor* (vid. pág. 6).

ANNAND, Ritchie, 2015. What is a plugin architecture? (Vid. pág. 33).

AROCA, Esmeralda Díaz, 2012. Wikitude: el mundo en realidad aumentada (vid. pág. 18).

COMPUTACIONAL, Programación, 2005. Visual C Sharp (vid. pág. 19).

CRUZ, Andrés, 2014. Realidad Aumentada con Vuforia (vid. pág. 17).

DAVID, 2013. Game Dev & Programming (vid. págs. 14, 15).

EDGAR, 2016. 10 razones para usar Git como software de control de versiones (vid. pág. 21).

E.GAMMA; HELM, R.; JOHNSON, R. y VLISSIDES, J., 1995. *Design Patterns* (vid. pág. 37).

FAVIERI, Adriana, 2008. Introducción a los cuaterniones (vid. pág. 15).

GALLEGO, Juana I., 2015. Los sensores de los celulares. En: *Los sensores de los celulares. Como influye en la experiencia del usuario la medición de variables físicas* (vid. págs. 7, 9-11).

GIRONÉS, Jesús Tomás, 2012. Sensores en dispositivos móviles. En: *Sensores en dispositivos móviles. El Gran Libro de Android* (vid. pág. 8).

GIROSCOPIO, 2017. Como funciona un giroscopio. Url: http://como-funciona.co/un-giroscopio/ (vid. pág. 12).

GOEBEL, Rainer, 2017. Spatial Transformation Matrices (vid. pág. 16).

HARGREAVES, Shawn, 2001. 31 Rutinas matemáticas para usar cuaterniones (vid. pág. 24).

JOSKOWICZ, José, 2008. Metodología XP (vid. pág. 21).

JUAN DE URRAZA, Carlos Giménez y, 2009. Acelerómetro (vid. pág. 7).

LETELIER, Patricio, 2006. Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP) (vid. pág. 21).

MARÍA MERINO, Julián Pérez Porto y, 2013. *Definición de plugin* (vid. pág. 11).

MARTTILA, Mika, 2015. Full-Tilt. Url: https://github.com/adtile/Full-Tilt (vid. pág. 17).

MENDOZA, 2010. Sección de Temas de Matemáticas (vid. pág. 16).

MILETTE, Greg y STROUD, Adam, 2010. *Professional Android Sensor Programming* (vid. pág. 13).

MONTOYA, Veronica, 2012. IMPLEMENTACION DE SOFTWARE (vid. pág. 39).

POLO, Ignacio Urbina, 2011. Orientación espacial: productos y territorios (vid. pág. 12).

PRESSMAN, Roger S., 2002. *Ingeniería de software. Un enfoque práctico Séptima edición*. Ed. por VÁZQUEZ, Pablo Roig (vid. págs. 41, 42).

PRIETO, Félix, 2009. Patrones de diseño (vid. pág. 37).

RIVERA, Nicolas, 2012. Así funciona el acelerómetro de tu smartphone (vid. pág. 7).

ROSE, William C., 2014. Rotation Matrices (vid. pág. 16).

SIERRA, Carlos Daniel, 2007. Visual Paradigm For UML (vid. pág. 20).

STUDIO, Android, 2018. Conoce Android Studio. Url: https://developer.android.com/studio/intro/?hl=es-419 (vid. pág. 20).

ULMEHER, Juan Manzano, 2015. Análisis de Android, el sistema operativo para móviles de Google. *Ibertronica*. Url: https://www.ibertronica.es/blog/tutoriales/android-sistema-operativo/ (vid. pág. 1).

UNITY, Documentación de, 2016. Plugins (vid. pág. 27).

WEI, Miao, 2013. Desarrollo de aplicaciones de sensores en teléfonos y tabletas Android basados en el procesador Intel Atom (vid. págs. 14, 24).

YEEPLY, 2014. Desarrollo de juegos con Unity 3D ¿Cómo funciona esta herramienta? (Vid. pág. 20).

Generado con LaTeX: 14 de junio de 2018: 10:45am

Apéndices

Implementación del método de corrección del sensor de orientación (Transformación de matriz de rotación a *quaternion*).

```
public float[] getQuaternion (float xx, float xy, float xz, float yx,
float yy, float yz, float zx, float zy, float zz) {
    // the trace is the sum of the diagonal elements; see
    // http://mathworld.wolfram.com/MatrixTrace.html
    final float t = xx + yy + zz;

    float x,y,z,w;

    // we protect the division by s by ensuring that s>=1
    if (t >= 0) { // |w| >= .5
        float s = (float)Math.sqrt(t + 1); // |s|>=1 ...
        w = 0.5f * s;
        s = 0.5f / s; // so this division isn't bad
        x = (zy - yz) * s;
        y = (xz - zx) * s;
        z = (yx - xy) * s;
    } else if ((xx > yy) && (xx > zz)) {
    float s = (float)Math.sqrt(1.0 + xx - yy - zz); // |s|>=1
        x = s * 0.5f; // |x| >= .5
        s = 0.5f / s;
        y = (yx + xy) * s;
        z = (xz + zx) * s;
        w = (zy - yz) * s;
    } else if (yy > zz) {
    float s = (float)Math.sqrt(1.0 + yy - xx - zz); // |s|>=1
        y = s * 0.5f; // |y| >= .5
        s = 0.5f / s;
        x = (yx + xy) * s;
        z = (zy + yz) * s;
        w = (xz - zx) * s;
    } else {
    float s = (float)Math.sqrt(1.0 + zz - xx - yy); // |s|>=1
        z = s * 0.5f; // |z| >= .5
        s = 0.5f / s;
        x = (xz + zx) * s;
        y = (zy + yz) * s;
        w = (yx - xy) * s;
    }
    return new float[]{x,y,z,w};
}
```

Aplicación para comprobar la información de los sensores en un dispositivo móvil.