



Asignatura: Desarrollo de Video Juegos

Pedro Yuri Marquez Solis

Manual – Unidad 1

ucontinental.edu.pe



Índice

Introducción	3
Organización de la Asignatura	4
Unidades didácticas	4
Tiempo mínimo de estudio	4
UNIDAD 1:	5
Diagrama de organización	5
Tema n.º 1:	6
Sub tema 1.1	6
Tema n.º 2:	15
Sub tema 2.1	15
De la teoría a la práctica	32
Glosario de la Unidad 1	37
Bibliografía de la Unidad 1	38

Introducción

Desarrollo de Video Juegos es una asignatura teórico-**práctica**; su intención es proporcionar al estudiante las herramientas indispensables para generar un aprendizaje autónomo, permanente y significativo en torno al fascinante mundo de los Video Juegos.

Busca que como estudiante seas capaz de desarrollar videojuegos 2D y 3D empleando un motor gráfico moderno considerando atributos de jugabilidad y eficiencia en el uso de recursos. Configura un aprender a aprender a través del reconocimiento de las habilidades de aprendizaje y la utilización de estrategias, atendiendo las necesidades y características de las actividades personales y académicas que pueda afrontar en esta modalidad.

En general, los contenidos propuestos en el manual autoformativo se dividen en cuatro unidades: Introducción al desarrollo de videojuegos, Componentes del proyecto, Scripting, finalmente Inteligencia artificial y comercialización del videojuego

Para afrontar con éxito este aprendizaje, le recomendamos que realice una permanente lectura de estudio de los contenidos desarrollados y de los textos seleccionados que amplían o profundizan el tratamiento de la información, junto con la elaboración de resúmenes y una minuciosa investigación vía Internet. El desarrollo del manual se complementa con autoevaluaciones, que son una preparación para la prueba final de la asignatura.

Organice su tiempo para que obtenga buenos resultados; la clave está en encontrar el equilibrio entre sus actividades personales y las actividades que asuma como alumno. El estudio a distancia requiere constancia; por ello es necesario encontrar la motivación que le impulse a ser mejor cada día mejor.

El autor

Organización de la Asignatura

Resultado de aprendizaje de la asignatura
Al finalizar la asignatura, el estudiante será capaz de desarrollar videojuegos 2D y 3D empleando un motor gráfico moderno considerando atributos de jugabilidad y eficiencia en el uso de recursos.

Unidades didácticas

UNIDAD 1	UNIDAD 2	UNIDAD 3	UNIDAD 4
Introducción al desarrollo de videojuegos.	Componentes del proyecto	Scripting	Inteligencia artificial y comercialización del videojuego
Resultado de aprendizaje Al finalizar la unidad, el estudiante será capaz de plantear el proyecto empleando Unity3d y esquemas de storyboarding	Resultado de aprendizaje Al finalizar la unidad, el estudiante será capaz de crear animaciones empleando diversos recursos y efectos de cámara.	Resultado de aprendizaje Al finalizar la unidad, el estudiante será capaz de controlar los elementos del videojuego y optimiza la jugabilidad e interacción con el usuario.	Resultado de aprendizaje Al finalizar la unidad, el estudiante será capaz de implementar la funcionalidad necesaria para que el CPU se enfrente al usuario, genera el ejecutable y lo publica en alguna tienda online especializada

Tiempo mínimo de estudio

UNIDAD 1	UNIDAD 2	UNIDAD 3	UNIDAD 4
16 horas	16 horas	16 horas	16 horas

UNIDAD 1:

Introducción al desarrollo de videojuegos

Diagrama de organización

Esquema que sintetice los contenidos a abordar y la relación entre los mismos

Tema n.º 1: Introducción al desarrollo de videojuegos

Con el transcurrir de la década del 2010 la industria de Video Juegos se ha afianzado, al punto de llegar a convertirse y superar a la industria musical y cinematográfica. Las cifras económicas que maneja llegan a superar los 20000 millones de dólares, compuesta básicamente por el diseño y desarrollo de los Video Juegos.

Ha quedado muy atrás la era del desarrollo sobre hardware, ahora desde una simple oficina que posea un hardware aceptable podemos desarrollar Video Juegos incluso aplicaciones de realidad virtual que antes eran exclusivos de grandes empresas o universidades.

Además, los métodos y técnicas han evolucionado a tal punto que la jugabilidad y calidad de los gráficos logran que cualquier persona pueda fácilmente caer en el atractivo de los Video Juegos.

Sub tema 1.1 Historia

Al finalizar la segunda guerra mundial los países vencedores iniciaron una carrera tecnológica, orientada a crear las más veloces computadoras basadas en tubos al vacío, pero fundamentalmente eran de programación general. (Roy, 2018)

Estos primeros juegos eran básicamente experimentos académicos, recién en la década de los 70 se tornan en productos comerciales.

1950-1951: Programming a Computer for Playing Chess

Durante la Segunda Guerra Mundial, **Alan Turing** y **Claude Shannon** trabajaron para descifrar los códigos secretos usados del ejército nazi con la máquina Enigma fruto de este trabajo establecieron las bases de la teoría de la computación, concluyendo que la Inteligencia artificial era el campo de investigación más importante en un futuro.

Consecuentemente en marzo del 1950 **Claude Shannon** presentó el artículo llamado "Programming a Computer for Playing Chess" donde se especificaban los primeras algoritmos y técnicas para un programa de ajedrez, las restricciones tecnológicas de potencia de procesamiento no permitieron su ejecución hasta noviembre de 1951, cuando se escribió el programa en la computadora Ferranti Mark I, la primera computadora electrónica comercial.



Imagen del terminal del Ferranti Mark I

Fuente: <http://www.otakufreaks.com/wp-content/uploads/2011/08/HDLV-Ferranti-Mark-1.jpg>

Con este programa, se sentaron las bases prácticas para la creación de los programas de ajedrez que se siguen aplicando hoy en día, al ser un juego simulado no tenía interfaz gráfica por lo que no se le puede considerar como el primer videojuego.

1952: OXO

Alexander Shaffo Douglas presentó en su tesis de doctorado de la **Universidad de Cambridge** una versión computarizada del juego tres en raya, este programa permitía enfrentar a un jugador humano contra la máquina **EDSAC (Electronic Delay Storage Automatic Calculator)**, **OXO** no tuvo popularidad ya que solo podía jugar en la **EDSAC**.

OXO es el primer juego de computadora en usar una pantalla gráfica, por lo que al mostrar gráficos, tiene todos los requisitos para ser considerarlo como el primer videojuego de la historia

1958: Tennis for Two

Considerado por muchos como el primer videojuego de la historia, fue creado por el físico **William Higinbotham**, el juego simulaba un partido de tenis a través de un osciloscopio el cual se utilizaba para calcular trayectorias de misiles para que cuando la gente visitara Brookhaven National Laboratory no se aburriera. Usaba una perspectiva lateral en la que se podía observar una línea horizontal que representa el campo de juego y otra pequeña vertical en el centro del campo que hacía de red. La pelota de tenis simulaba el efecto de gravedad.

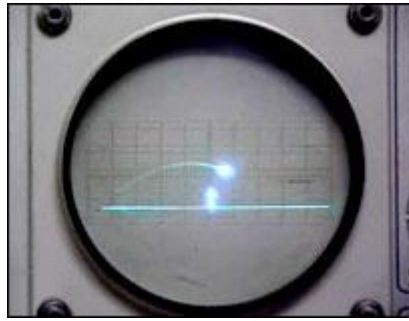


Imagen del Tennis for Two

Fuente: https://en.wikipedia.org/wiki/Tennis_for_Two

En 1961, apareció Spacewar, considerado por otros como el primer videojuego de computador en la historia, permitía jugar dos personas controlando una nave espacial con la finalidad de destruir al rival



Imagen de Spacewar! (1962, Steve Russell)

Fuente:

<http://malditosnerds.com/malditosBackend/web/uploads/top/20160331165353.jpeg>

Luego de Spacewar! llegarían juegos como el primer simulador de beisbol (1965), Ping Pong (1967), Space Travel, Lunar Lander y Hamurabi (todos en 1969), Highnoon (1970), Galaxy Game y Star Trek (1971) y finalmente el mítico Pong de Atari en 1972.

Artículo: Historia de los videojuegos: El origen y los inicios
Enlace: <http://www.otakufreaks.com/historia-de-los-videojuegos-el-origen-y-los-inicios>
Fecha de publicación: 10 de octubre de 2011
Autor: Ramon Tomàs Piqué

En el libro Desarrollo de Video Juegos un enfoque práctico se menciona que:

“La evolución de la industria de los videojuegos ha estado ligada a una serie de hitos, determinados particularmente por juegos que han marcado un antes y un después, o por fenómenos sociales que han afectado de manera directa a dicha industria. juegos que han marcado un antes y un después, o por fenómenos sociales que han afectado de manera directa a dicha industria.

Juegos como Doom, Quake, Final Fantasy, Zelda, Tekken , Gran Turismo, Metal Gear,

The Sims o World of Warcraft, entre otros, han marcado tendencia y han contribuido de manera significativa al desarrollo de videojuegos en distintos géneros. Por otra parte, y de manera complementaria a la aparición de estas obras de arte, la propia evolución de la informática ha posibilitado la vertiginosa evolución del desarrollo de videojuegos. Algunos hitos clave son por ejemplo el uso de la tecnología poligonal en 3D"

Sub tema 1.2 Clasificación

Los Video Juegos poseen distintas clasificaciones, una de las más aceptadas debido a que se relaciona directamente con el motor de juegos es la siguiente:

FPS (first person shooter): Wolfenstein 3D, Doom, Unreal, Quake, Half Life, y posteriormente Halo, Call of Duty, Team Fortress, Far Cry o Battlefield. El usuario normalmente controla a un personaje con una vista en primera persona a lo largo de escenarios normalmente interiores. Requieren de desarrollo complejo, debido a la alta inmersión del usuario, los detalles de imágenes, así como un tiempo de respuesta muy corto.

Juegos en tercera persona: Resident Evil, Metal Gear, Gears, of War o Uncharted, entre otros. En estos juegos el usuario tiene el control de un personaje cuyas acciones se aprecian completamente desde la cámara se centran en la animación del personaje, destacando sus movimientos y habilidades.

Juegos de lucha: Virtua Fighter , Street Fighter, Tekken , o Soul Calibur. Los jugadores compiten para ganar un determinado número de peleas minando la vida del jugador contrario, se desarrollan normalmente en escenarios tridimensionales pero el movimiento de los luchadores está limitado a dos dimensiones (de scroll lateral).

Conducción: Gran Turismo, y arcade , como por ejemplo Ridge Racer o Wipe Out.

El usuario controla un vehículo que rivaliza con más adversarios virtuales o reales para llegar a la meta en primera posición. Se tienen en dos tipos simuladores; representan con fidelidad el comportamiento del vehículo y su interacción con el escenario, y los arcade; se centran más en la jugabilidad para que cualquier tipo de usuario.

Juegos De Estrategia Tiempo real o RTS (Real-Time Strategy)) y por turnos (turn-based strategy, Otro género tradicional son los juegos de estrategia, normalmente clasificado

Warcraft, Command & Conquer , Comandos, Age of Empires Starcraft , Mantienen la cámara con una perspectiva isométrica, normalmente fija, de manera que el jugador tiene una visión más o menos completa del escenario, ya sea 2D o 3D, emplean un gran número de unidades virtuales desplegadas en el mapa.

MMOG(Massively Multipla-yer Online Game). World of Warcraft. posibilidad de jugar con un gran número de jugadores reales al mismo tiempo, del orden de cientos o miles de jugadores.

Sub tema 1.3 Referentes de los Video Juegos

Desde la aparición de los Video Juegos han surgido aquellos que han marcado significativamente el desarrollo de esta industria, cada uno mencionado ha aportado alguna técnica, un género o una mecánica que los jugadores han marcado como un estándar de los Video Juegos.

Minecraft

Creado por el estudio Mojang, aporta el ideal que "el único límite es tu imaginación", emplea cubos formados por píxel gordos, ofrece crear lo que uno quiera convirtiéndolo en uno de los más vistos en YouTube y uno de los más jugados en el mundo.

Resident Evil

Sentó las bases del genero conocido como survival horror, la idea es agobiar al jugador con la sensación de vulnerabilidad e indefensión por los peligros que va encontrando. Generó otros juegos como Dino Crisis, Parasite Eve o Silent Hill.

Metal Gear Solid

Juego por excelencia de sigilo, impulsó un antes y un después debido al salto grafico en lo que a juegos de acción en 3d con perspectiva en tercera persona.

Super Mario Bros.

Juego de plataforma por excelencia, fue un fenómeno en la consola de Nintendo NES, su diseño de niveles y mecánicas sentaron los estándares del género de plataformas.

DOOM

Desarrollada por id Software, uno de los primeros juegos en exigir los FPS(frame per second) debido a su ritmo y violencia representada , introdujo novedosas ideas tanto técnicas como a nivel de mecánicas.

The Legend of Zelda: Ocarina of Time

Cambió y estandarizó las mecánicas de los videojuegos de aventuras, uso y gestión de ítems, agregó posibilidades de exploración, además de misiones secundarias y minijuegos que a su vez competían en diversión con la historia principal.

PACMAN

Aportó una sencillez excepcional en la mecánica de los juegos, la idea era conseguir que el personaje principal lograra comer 240 puntos colocados en un laberinto, se perdía cuando era atrapado por alguno de los 4 fantasmas, se llegaron a vender 100000 máquinas de PacMan sólo en Estados Unidos.

Sub tema 1.4 Introducción a las técnicas de los Video Juegos

Al ser el computador y humano entes diferentes, enfrentarlos en el territorio humano es dificultoso, por lo que se tienen 4 técnicas para lograr que el reto al humano se más justo y retador. **(Adrigm, 2018)**

Recursos Vastos: la ventaja del jugador humano en inteligencia se contrarresta por las ventajas materiales de la computadora.

Es la técnica más utilizada para balancear un juego. Se provee a la computadora de una inmensa cantidad de recursos que utiliza torpemente; como un gran número de oponentes con inteligencia rudimentaria. Entre los juegos que utilizan esta táctica están: Space Invaders, Missile Command, Asteroids, Centipede, y Tempest , algo parecido es brindar a la computadora de un pequeño número de oponentes más poderosos que las unidades del jugador humano, como por ejemplo los supertanks en Battlezone.

Este enfoque tiene dos beneficios.

- Brinda a la historia del enfrentamiento entre humano y computadora un aire de David contra Goliat. La mayoría de la gente prefiere ganar con una aparente desventaja que en igualdad de condiciones.
- Este enfoque es el más fácil de implementar, ya que proveer de IA a las unidades de la computadora puede ser difícil, pero repetir un proceso para muchas unidades de la computadora sólo necesita de algunos ciclos de reloj

Razonamientos Artificiales: se pueden crear rutinas adhoc rudimentariamente inteligentes.

Considerando que la IA no está aún muy desarrollada y aunque las técnicas de árbol producen jugadores o unidades aceptables en ajedrez, damas y Othello, se aplican Razonamientos artificiales, se han utilizado por ejemplo en Tanktics, Easter Front 1941, y Legionnaire, con diversos niveles de éxito. Esta estrategia exige gran esfuerzo de parte del diseñador, puesto que dichas rutinas ad-hoc deben ser razonables pero impredecibles, considerando por lo menos los siguientes atributos:

- Comportamiento razonable: La computadora debe evitar realizar movimientos tontos y obvios, por ejemplo no enviar sus unidades a abismos, no detenerse ante armas del jugador humano y otros. Un error común para implementarlo es enlistar todos los movimientos tontos y mediante código comprobar cada movimiento tonto y eliminarlo. Esta forma es incorrecta ya que la computadora

puede demostrar creatividad no anticipada en la estupidez de sus errores. Una mejor forma, pero más difícil solución es crear un algoritmo más general que obvie los movimientos más absurdos.

- Impredecible: El jugador humano jamás debe adivinar el comportamiento de las unidades del computador, porque esto elimina la ilusión de inteligencia y permitiría la victoria humana de una forma muy fácil. La acción del computador es razonable, pero impredecible.
- Uso de algoritmos “particulares: Considerando que muchos CPUs no tienen ni la potencia ni tecnología para anticipar los movimientos humanos interactivamente, debemos apoyarnos en guías más primitivas, el uso de algoritmos “particulares” que ponen énfasis en elementos únicos de todo el patrón del juego, por ejemplo, en juegos de guerra, algoritmos del tipo “determina al enemigo más cercano y dispárale” son particulares y demuestran comportamiento predecible. Un mejor algoritmo podría ser “determina la unidad enemiga que ofrece la mejor combinación de amenaza y vulnerabilidad (basado en distancia, actividad, dirección, distancia a otras unidades amigas, protección, y visibilidad); dispara a la unidad si la probabilidad de matarla sobrepasa la probabilidad de morir”.

Una forma de implementarlos es utilizar sistemas de punto, análisis de campo, y cambios en la estructura del juego. Se establece un sistema de punto para cuantificar con puntaje cada movimiento posible. Esta es una técnica importante para muchos sistemas de inteligencia artificial. Muchísima planeación debe ser utilizada para el sistema de punto. En un hipotético juego de tanques estar en la cima de una colina da buen puntaje, pero moverse a un camino también es bueno. ¿Cuál es mejor? Desafortunadamente es difícil de contestar, la única alternativa es la experiencia, un conocimiento íntimo de la situación representada, análisis exhaustivo, y mucha experimentación.

- Uso de análisis de campo: se aplica a juegos que involucren relaciones espaciales. En dichos juegos el humano depende de reconocer patrones para analizar posiciones y planear movimientos. Por ejemplo, en un juego de guerra se determinan campos de seguridad y de peligro que le dicen a una unidad cuanta seguridad y riesgo enfrenta. El peligro es calculado al sumar los coeficientes de fuerza de las unidades enemigas dividido entre sus alcances; de esta manera, unidades grandes y cercanas son muy peligrosas y unidades pequeñas y distantes son poca amenaza. Un cálculo similar con unidades

aliadas resulta en un factor de seguridad.

- Quitar o modificar elementos difíciles de implementar: si un elemento del juego no es adaptable fácilmente para los cálculos artificiales entonces es mejor quitarlo. Por ejemplo, en juego de tanques se tenía un problema con los lagos de forma cóncava, a pesar del mucho tiempo desperdiciado buscando la mejor solución no se obtuvo por lo que se tuvo que borrar los lagos cóncavos del mapa.
- Movimientos coordinados sin atascos: Otro problema especial es la coordinación de movimientos de muchas unidades diferentes bajo el control de la computadora. ¿Cómo lograr que la computadora mueva muchas unidades de manera coordinada y que no se desarrollen embotellamientos?. Una manera es utilizar un sistema de planeación secuencial junto con una prueba sencilla para la posición de otras unidades. Así, la unidad #1 se mueve primero, luego #2, luego #3, evitando colisiones, este sistema crea aún más embotellamientos, una mejor manera es iniciar con un arreglo de posiciones reales de todas las unidades de computadora y un arreglo con las posiciones virtuales forman la base de los movimientos hechos por las unidades de computadora. Esta técnica debe ser útil para coordinar movimientos de muchas unidades y prevenir embotellamientos.

Información Limitada: La información limitada provee una bonificación: puede tentar la imaginación del jugador al sugerir sin confirmar, pero solo tiene éxito cuando las limitaciones en la información son artísticamente elegidas.

Ritmo del juego: Debido a que una computadora es mucho más rápida en realizar cálculos simples que un humano, si el ritmo es lo suficientemente rápido, el humano no tendrá tiempo de aplicar sus habilidades superiores de procesamiento y será aturdido. Esta es una técnica muy fácil de aplicar, por lo que no sorprende que sea ampliamente utilizada por diseñadores de juegos de habilidad y acción, pero contrariamente no se recomienda salvo mediante la limitación al jugador humano del tiempo que necesita para invertir una porción más larga de sí mismo en el juego.

Estas cuatro técnicas para balancear juegos de computadora jamás son usadas aisladamente; cada juego utiliza alguna combinación de las 4. La mayoría de los juegos se basan principalmente en ritmo y cantidad para balancear, con muy poca inteligencia o información limitada. No hay razón para que un juego no pueda utilizar las 4 técnicas; ciertamente, esto debería hacer al juego más exitoso, ya que, al usar pequeñas cantidades de cada método, el juego no tendrá que forzar las limitaciones de cada uno. El diseñador debe decidir el balance apropiado de cada uno de los

objetivos del juego en particular.

Tema n.º 2: Unity3D

Unity 3D es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X, Linux. Permite la creación de juegos para múltiples plataformas a partir de un único proyecto incluyendo el desarrollo de juegos para consolas; PlayStation, Xbox y Wii, escritorio; Linux, PC y Mac, navegador, móviles y tablets; iOS, Android, Windows Phone y BlackBerry). Unity tiene dos versiones: Unity Professional (pro) y Unity Persona.

Todo este material se ha elaborado considerando la versión de Unity 3D 2017.2

Sub tema 2.1 Funcionalidades básicas de Unity 3D

Unity 3D es un motor de videojuegos, por ende, posee un conjunto de herramientas que realizan cálculos geométricos y físicos que se emplean en los videojuegos. Integrado por un simulador ágil en tiempo real que reproduce las características de los escenarios en los que transcurren los videojuegos. De esta manera el equipo de desarrollo de un videojuego se puede concentrar en el contenido del Video Juego y no en la resolución de problemas informáticos. Finalmente se logra que usuarios con pocos conocimientos de programación puedan manejar con facilidad el motor y creen Video Juegos.

Unity permite obtener un resultado de máxima calidad con un mínimo de esfuerzo. Unity se proporciona en dos versiones:

Versión profesional: se puede adquirir previo pago

Versión libre: completamente gratuita que se puede descargar en la página Web de Unity, posee menos funcionalidades pero permite la creación de videojuegos de buena calidad.

Unity es una aplicación 3D en tiempo real y multimedia además de ser motor 3D y físico utilizado para la creación de juegos en red, de animación en tiempo real, de contenido interactivo compuesto por audio, video y objetos 3D. El motor de Unity 3D no permite modelización de objetos, pero permite insertar escenas que soportan iluminación, terrenos, cámaras, texturas.

Funcionalidades:

Permite importación de numerosos formatos 3D como son 3ds Max, Maya, Cinema 4D, Cheetah3D y Softimage, Blender, Modo, ZBrush, FBX, además de recursos de texturas en formatos Photoshop, PNG, TIFF, audios y videos, además es posible optimizar estos

recursos mediante la aplicación de filtros.

Unity 3D, es compatible con diversas API graficas como son Direct3D, OpenGL y Wii, además es compatible con QuickTime y utiliza internamente el formato Ogg Vorbis para audio.

En Unity, el juego se construye mediante el editor y un lenguaje de scripts que pueden ser JavaScript, C# y un dialecto de Python denominado Boo .

En Unity 3D la estructura del proyecto se define mediante escenas que representan alguna parte del juego. Se incluye un editor de terrenos desde cero, permite esculpir la geometría del terreno, su texturización y la inclusión de elementos 3D importados desde aplicaciones 3D o ya predefinidos en Unity.

Los recursos para un videojuego, se puede acceder desde el Asset Store donde existe multitud de recursos gratuitos y de pago.

Dispone de una interfaz de desarrollo muy bien definida e intuitiva que permite crear rápidamente mini-juegos.

Sub tema 2.2 Entorno de trabajo 2d y 3d.

Luego de instalar Unity 3D e iniciar el programa se aprecia una ventana que permite elegir por un Nuevo proyecto (1), aperturar un proyecto(2), elegir la ubicación del proyecto en Disco(3) o en la Nube(4)



Fig Nro 1. Pantalla inicial de Unity 3D

Si se eligió crear un proyecto nuevo se verá una nueva ventana donde se debe elegir entre un proyecto en 3D ; tres dimensiones o 2D; dos dimensiones. Se ingresa el nombre del Proyecto (1), luego elige el tipo (2), indicar la ubicación del archivo (3) y finalmente

clic al botón Create Project (4).

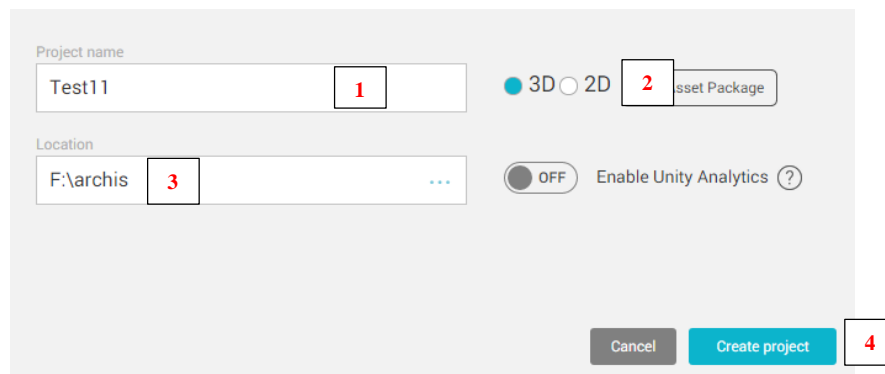


Fig Nro 2. Pantalla Inicial de Unity 3D

En la interfaz de Unity 3D existen principalmente 5 áreas 3D, numeradas en la imagen de abajo.

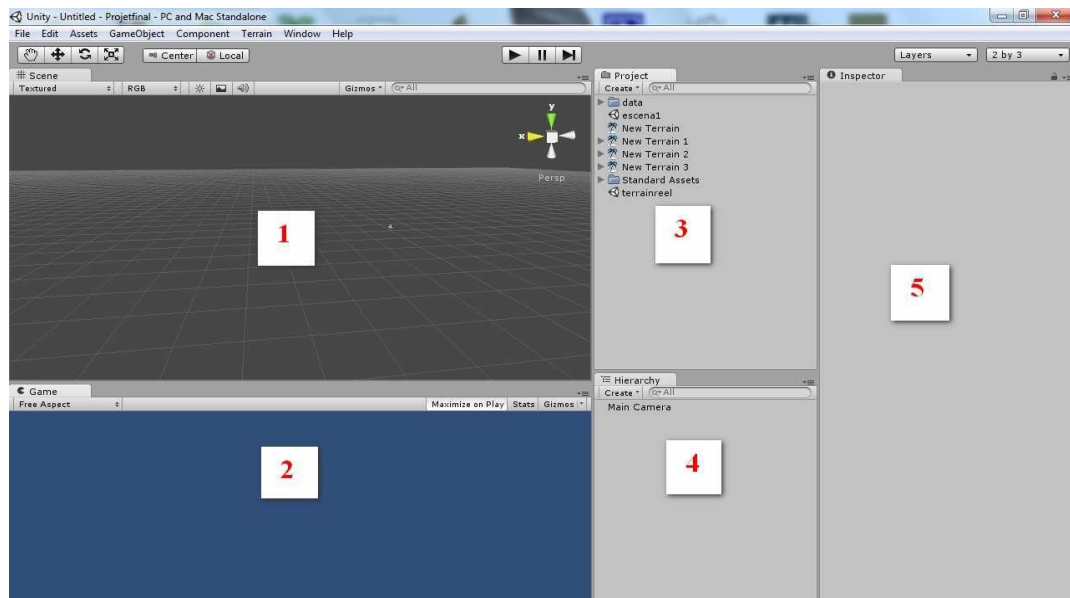


Fig Nro 3. Pantalla principal de Unity 3D

Elemento	Descripción
Ventana 1	ventana de escena , es el área de construcción, donde se va agregando visualmente cada escena del juego.
Ventana 2	vista de juego , muestra una pre-visualización del juego, que en cualquier momento puede reproducir el juego
Ventana 3	vista del proyecto , es la librería de assets para el juego. Se pueden importar objetos 3D de distintas aplicaciones a la librería, se pueden importar texturas y crear otros objetos como Scripts. Todos los assets del juego se almacenan aquí para que se puedan usar en el juego.
Ventana 4	vista de jerarquía , contiene todos los objetos de la escena actual.
Ventana 5	vista de inspector , permite seleccionar objetos y que se muestren sus propiedades y personalizar sus características.









Tool Bar:

Debajo del menú de Unity vienen definidos los botones de control tal y como se ve en

la imagen que sigue:



Fig Nro 4. Botones de manipulación

	<p>Este botón permite moverse alrededor de la vista de escena:</p> <ul style="list-style-type: none"> • ALT permite rotar. • COMMAND/CTRL permite hacer zoom. • SHIFT incrementa la velocidad de movimiento mientras se usa la herramienta.
	Permite mover los objetos seleccionados en la escena, el movimiento se puede realizar en los ejes X, Y e Z.
	Permite rotar los objetos seleccionados en la escena, la rotación se puede realizar en los ejes X, Y e Z.
	Permite escalar los objetos seleccionados en la escena. En este caso también se puede escalar en cualquiera de los ejes X, Y y Z.
	Reproduce el juego sin salir del editor. El primer botón permite lanzar el juego y visualizarlo, el segundo permite pausarlo y el último a la derecha permite saltar adelante en el juego. Se puede visualizar el videojuego en la vista de juego tal como está o maximizando la pantalla.
	Los objetos se pueden transformar en diferentes sistemas de coordenadas y desde diferentes puntos de pivote
	Es usado para para controlar qué objetos son representados por qué cámaras o iluminados por qué luces
	El menú desplegable Capas le permite volver rápidamente a varios diseños predeterminados o predefinidos.

Sub tema 2.3 Game objects:

En Unity 3D cada objeto del juego es un GameObject, los GameObjects no hacen nada por sí mismos. Requieren de propiedades antes de que puedan conformar un personaje, un ambiente, o un efecto especial. Pero cada uno de estos objetos hacen diferentes cosas

Siempre es útil tener algo en la escena cuando se va a navegar en aplicaciones 3D. Aunque importará la mayoría de los assets, encontrará muchos usos para los Objetos primitivos disponibles en el motor de juegos de Unity.

Para crear los objetos predefinidos en Unity 3D, se accede a GameObject > Create Other se despliega una lista de elementos que se muestra en la Fig. Nro 5:

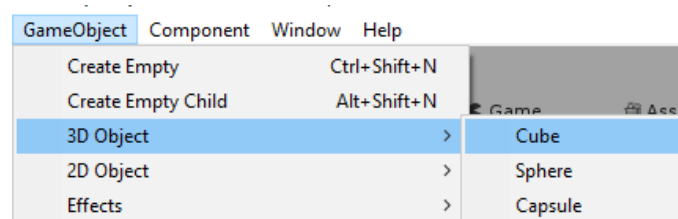


Fig Nro 5. Creando un game Object

Por ejemplo, se crea un cubo en el centro de la ventana de escena. También puede verlo como un pequeño cuadrado oscuro en la ventana del juego.

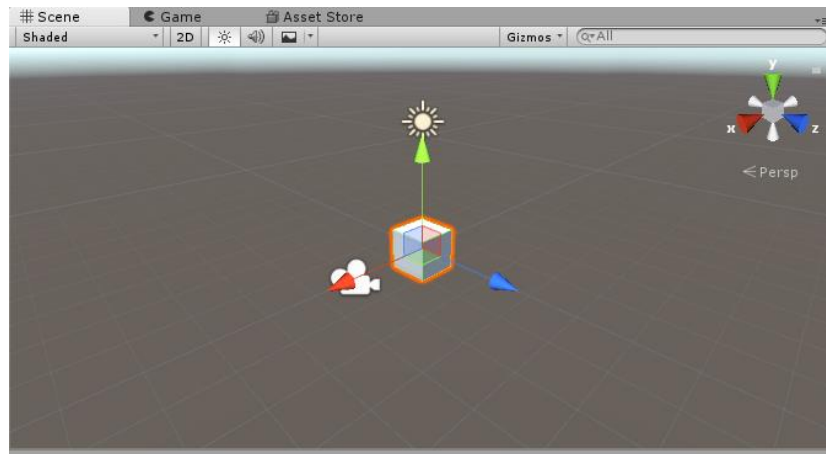


Fig Nro 6. Game Object de tipo Cube agregado

Al seleccionar uno de los “GameObject”, este aparece en las vistas de escena, la de juego y jerarquía. Además en la vista de inspector se pueden ver los atributos del objeto para la parametrización de este.

Escena

Las escenas contienen los Game Objects de su juego. Se pueden usar para crear un menú principal, niveles individuales, y cualquier otro elemento. Cada archivo de escena es como un nivel único. En cada escena, se colocan el ambiente, obstáculos, decoraciones, el diseño esencial y otros elementos de su juego.

Al crear un nuevo proyecto de Unity, su scene view muestra una nueva escena, inicialmente nombrada como escena *untitled* (sin título) y *unsaved* (sin guardar). La escena estará vacía al menos por objetos predeterminados - ya sea una cámara ortográfica, o una cámara perspectiva y una directional light, dependiendo si comenzó el proyecto en modo 2D o 3D.

Guardando Escenas

Para guardar la escena en la cual usted está actualmente trabajando, escoja File > Save Scene desde el menú, o presione Ctrl + S.

Las escenas se guardan como assets, a la carpeta de Assets de su proyecto. Por lo tanto, aparecen en la ventana del Proyecto, como cualquier otro asset.

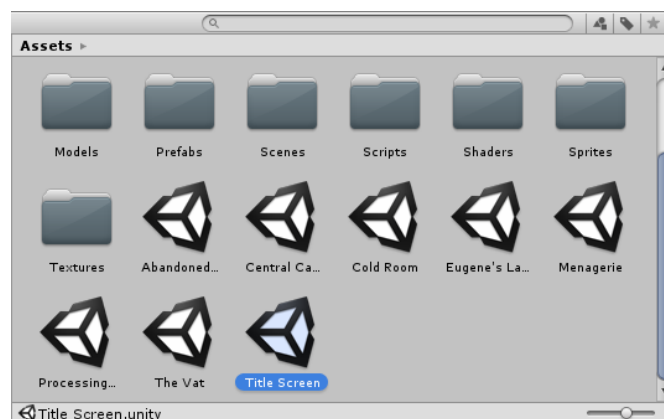


Fig Nro 7. Assets de escena, guardados y visibles en el project view

En la Fig. Nro 8 se muestran 4 diferentes Game Objects, el primero un personaje animado, el siguiente una luz, luego tenemos un árbol y finalmente una fuente de audio. Al ser los Game Objects contenedores pueden guardar las diferentes piezas que son requeridas para hacer un personaje, una luz, un árbol, un sonido, o lo que se requiera construir. Para entender estas piezas que son llamadas Components, se van agregar diferentes combinaciones de Components al GameObject.

Piense en un GameObject como una olla vacía de cocina, y los Components como los ingredientes diferentes que hacen su receta del Video Juego. Unity tiene varios tipos de componentes integrados, y usted también puede hacer sus propios componentes utilizando Scripts.

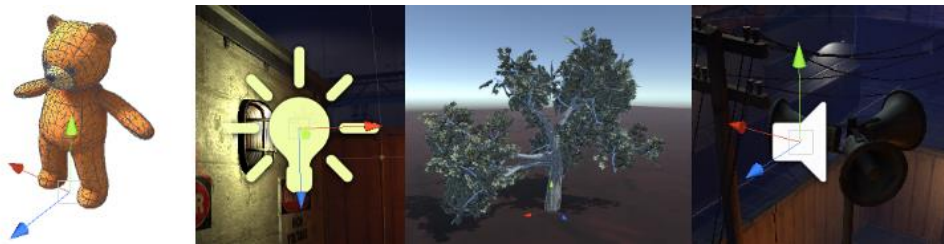


Fig Nro 8. Ejemplos de Game Objects

Recordemos que los Game Objects no logran nada por sí mismos, pero funcionan como contenedoras para Components, que implementan la verdadera funcionalidad. Por ejemplo, un objeto Light es creado al adjuntar un componente Light a un GameObject.

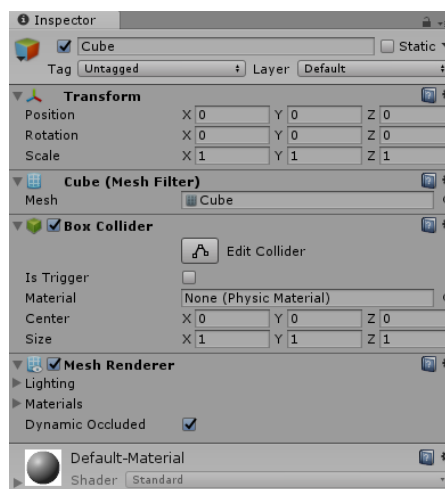
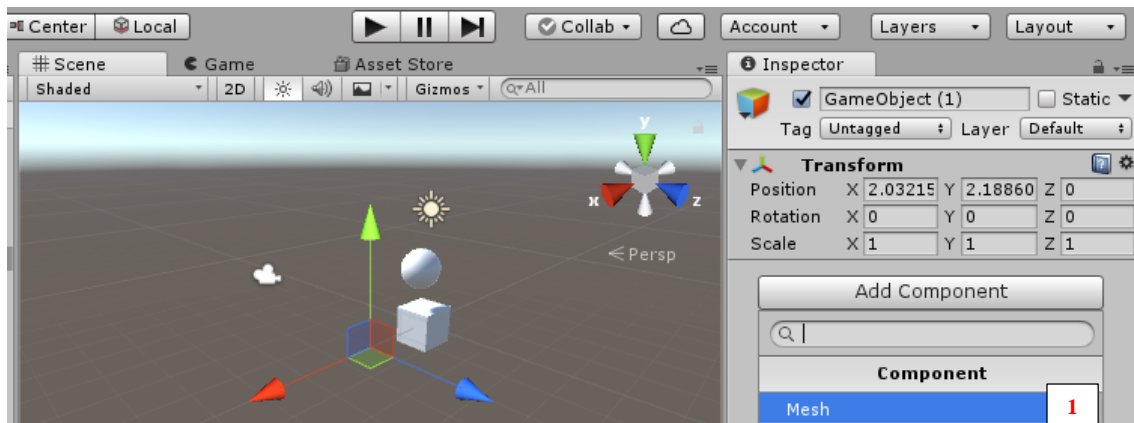


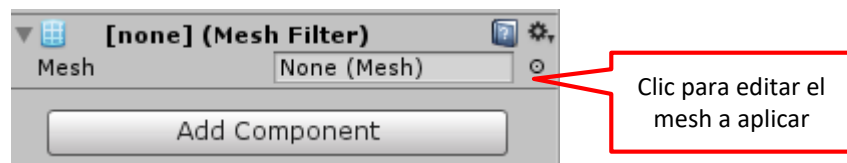
Fig Nro 9. Componentes de un Game Object

Adjuntando elementos a un Game Object:

Hemos agregado un Game Object vacío, al seleccionarlo se muestra en el inspector la opción Add Component, al efectuar clic muestra una lista de componentes a elegir, por ejemplo, un objeto Mesh(malla).



Luego de agregar la malla en la misma ventana es posible editar el filtro de malla.



Luego puede elegir el objeto de filtro, por ejemplo, en la Fig. Nro 10 se selecciona una esfera.

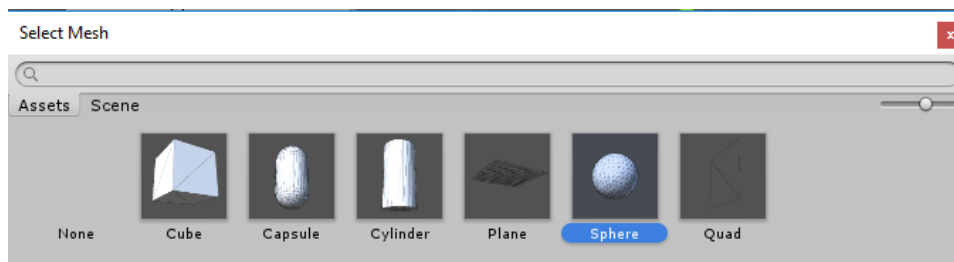


Fig Nro 10. Agregando malla de tipo Esfera

Ahora el Game Object tendrá la forma de esfera:

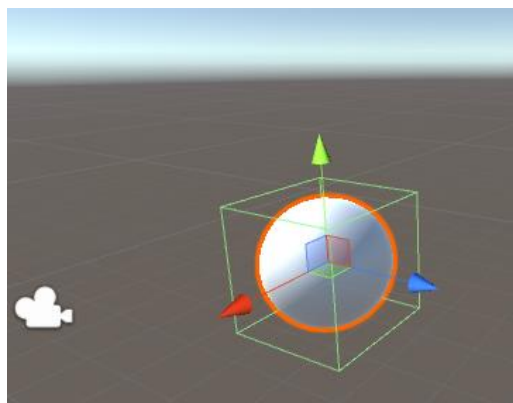


Fig Nro 11. Game Object con mesh esfera

Ahora puede agregar un asset, como para visualizar más fácilmente el efecto a aplicar. Siga la secuencia: Menu assets, opción importar y elija algún archivo bmp, jpg o png, por ejemplo, hemos tomado el logo de la Universidad Continental y desde la vista de assets se ha arrastrado hasta el Game Object.

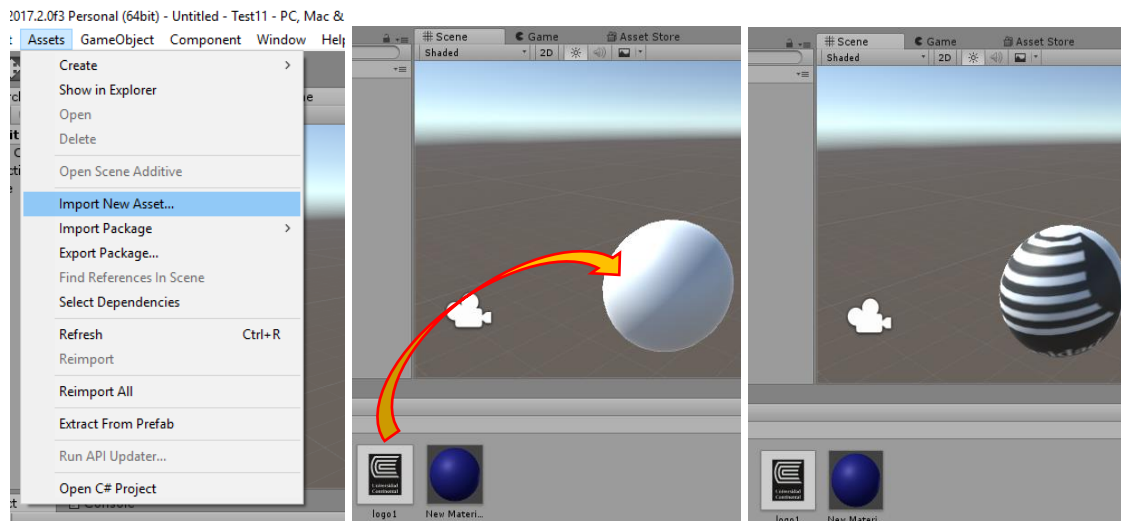


Fig Nro 12. Estableciendo un asset como shader

También podemos cambiar nuevamente el mesh del objeto,

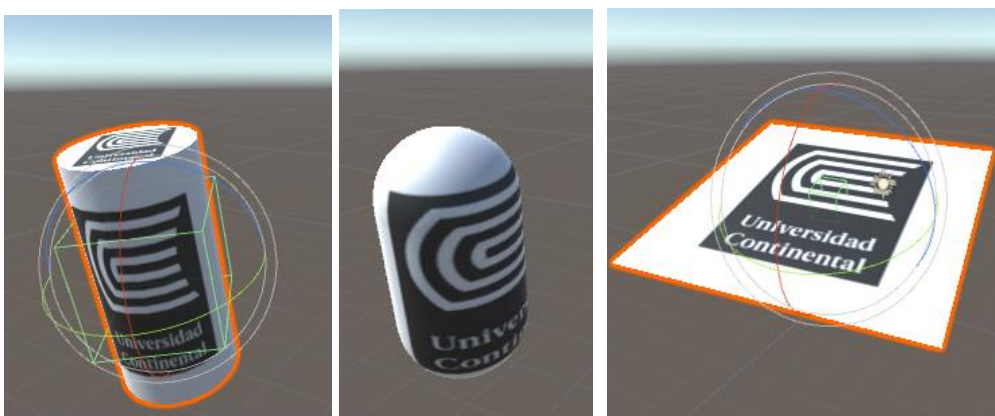


Fig Nro 13. Distintas mallas aplicadas a un Game Object

Sub tema 2.4 Introducción a la programación

Vamos a agregar elementos de programación a nuestros objetos, para ello seguimos la secuencia: menú assets → opción Create, opción C# Script, se genera un archivo con nombre temporal *NewBehaviourScript* el cual debe ser cambiado, por ejemplo le pondremos **script1**.

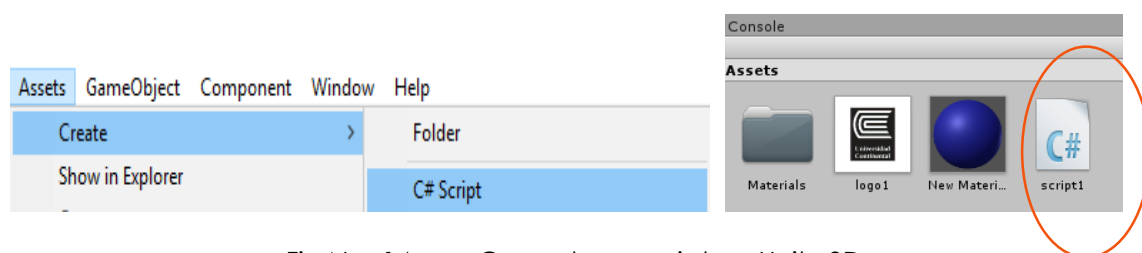


Fig Nro 14. Creando un script en Unity 3D.

Se activará Visual Studio 2017 con la pantalla de código como se muestra en la Fig. Nro 15. Crea por

defecto una clase con el mismo nombre del script, en este caso **Script1**. En esta clase se distinguen dos zonas principales:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class script1 : MonoBehaviour {
```

```
    // Use this for initialization
```

```
    void Start () {
```

Se usa la primera vez que inicia el script

```
    // Update is called once per frame
```

```
    void Update () {
```

Se usa repetitivamente como un loop.

```
}
```

Ahora vamos a agregar el código siguiente:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class script1 : MonoBehaviour {
```

```
    // Use this for initialization
```

```
    void Start () {
        Debug.Log("Continental");
    }
```

```
    // Update is called once per frame
```

```
    void Update () {
        Debug.Log("Continental en bucle");
    }
```

```
}
```

Luego debemos asociar el script como un component de un Game Object, en este caso lo arrastraremos sobre el objeto **Main Camera**, si observamos en la vista inspector ahora posee un nuevo componente, el script1.

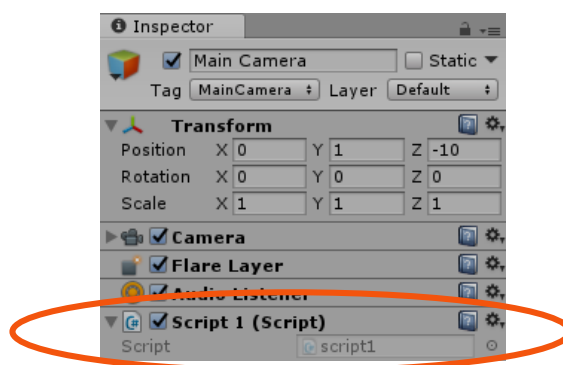


Fig Nro 15.

Ahora activamos la vista Console y ejecutamos efectuando clic en el botón de control de simular el juego. Podremos apreciar en la vista de consola los efectos de la ejecución del script.

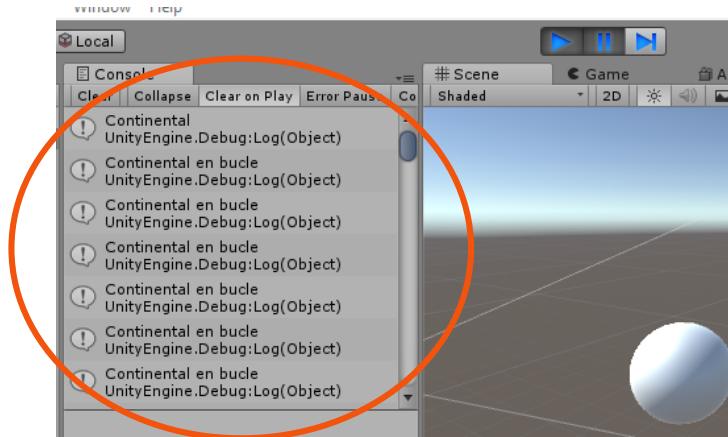


Fig Nro 16. Ejecutando el script1

Podemos mejorar el script1 agregándole una función de rotado al objeto, para ello agregue un nuevo script y nómbrelo Rotar.

```
public class Rotar : MonoBehaviour {
    void Start () {
    }
    void Update () {
        transform.Rotate(0, 5, 0);
    }
}
```

Ahora asocie el script al objeto del logo, a continuación ejecute el programa, verá que la esfera rota sobre su eje, esto debido a que la instrucción `transform.Rotate(x,y,z)` rota el objeto en los ejes que se le indique.

Prueba agregando la función de tiempo `deltaTime`, Cuando multiplicas por `Time.deltaTime` lo que haces es expresar: quiero mover este objeto 15 metros por segundo en vez de 15 metros por frame.

```
transform.Rotate(0,45*Time.deltaTime ,0);
transform.Translate(2 * Time.deltaTime, 0, 0);
```

Podrá observar que ahora el objeto gira lentamente sobre su eje, pero además también rota en una trayectoria.

Sub tema 2.5 Aplicación básica con Unity

Vamos a crear el clásico Pong en 3D, pero en su estructura básica, es decir que la bola pueda rebotar entre las paredes y las paletas, aún no vamos a asignar ningún color ni textura, ni tampoco el movimiento de las paletas. La idea es que tenga un panorama de cómo se insertan y articulan los elementos de un Video Juego. También se familiarizará con el sistema de coordenadas en 3D.

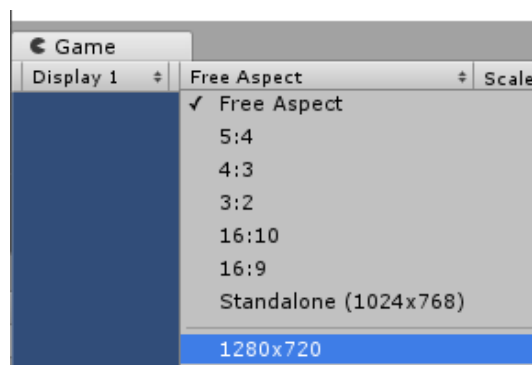
Iniciamos creando un nuevo proyecto en Unity en 3D.

Configuramos el objeto main camera con los siguientes atributos:

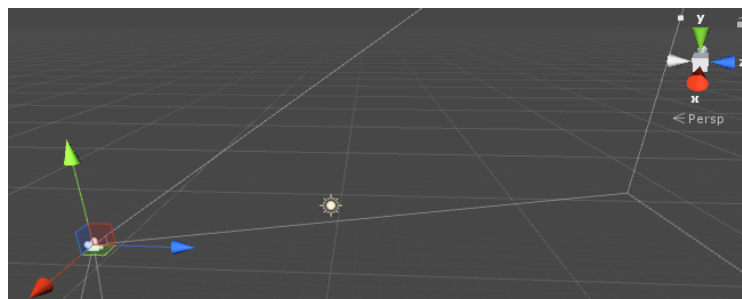
1. Color sólido para el cielo, esto permite apreciar rápidamente los cambios de los objetos a utilizar. También el valor de perspectiva en el valor 30.



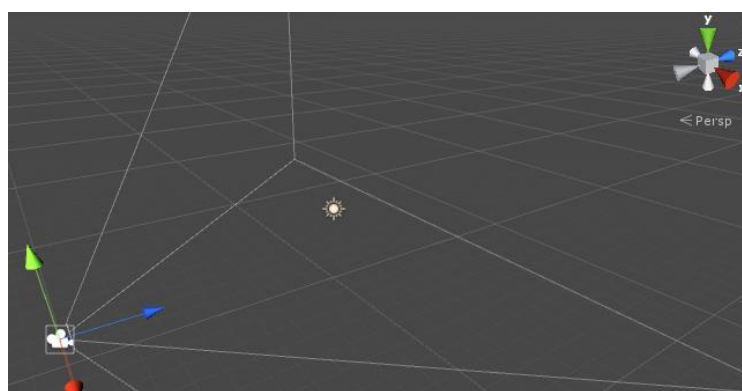
2. Definir en la ventana Game la resolución de 1280*720



3. En el objeto Scene, modificar el visor de perspectiva de tal modo que el eje Z quede hacia atrás de la cámara, ya que la bola se moverá en las dimensiones X e Y. Para ello mantengan presionada la tecla Alt

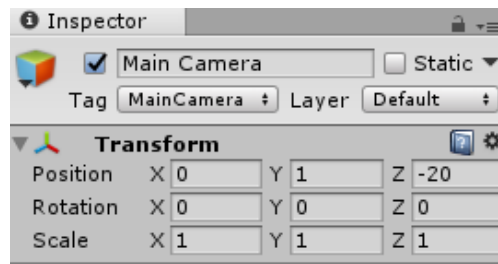


Posición inicial del
eje Z



Posición final del
eje Z

4. Agrega a la configuración de cámara en la posición Z= -20, para alejar la cámara del origen de coordenadas.



5. Ahora crearemos las paletas a las que denominaremos player1 y player2, agregamos un objeto 3D cube, y establecemos las siguientes propiedades:

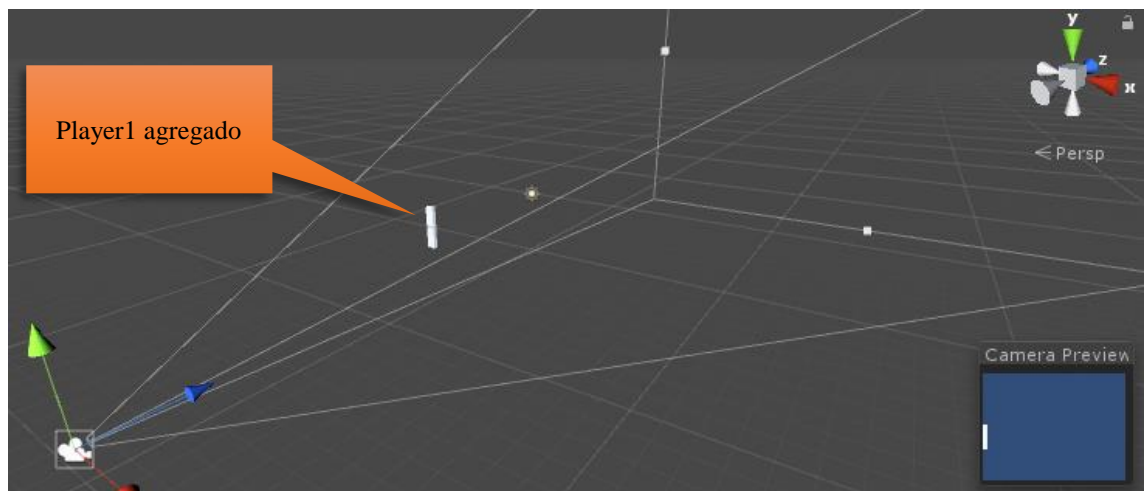
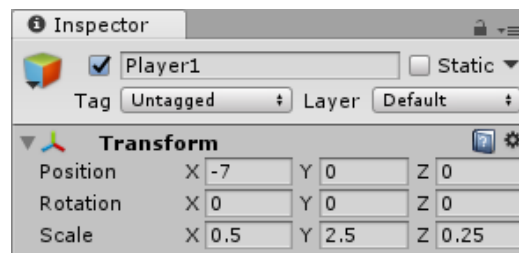
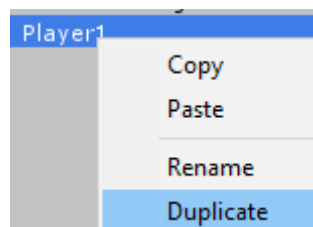
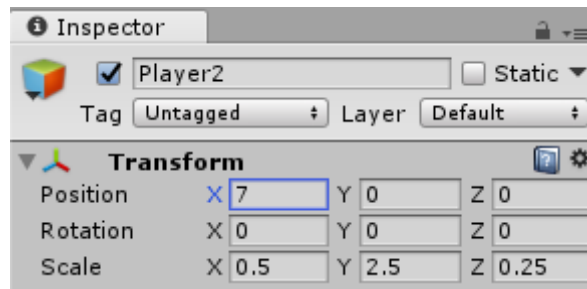


Fig Nro 17.

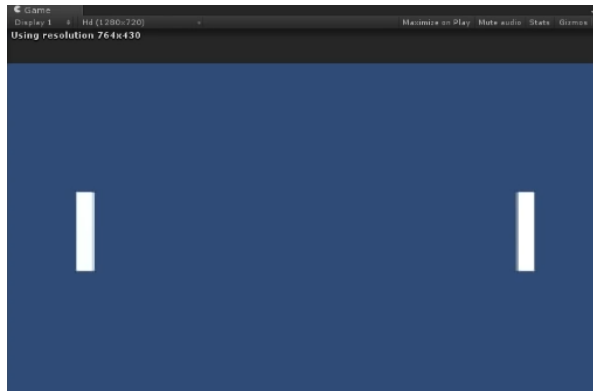
6. Con el mismo procedimiento agregue el Player 2 y establezca los siguientes valores, lo puedes hacer con duplicar objeto:



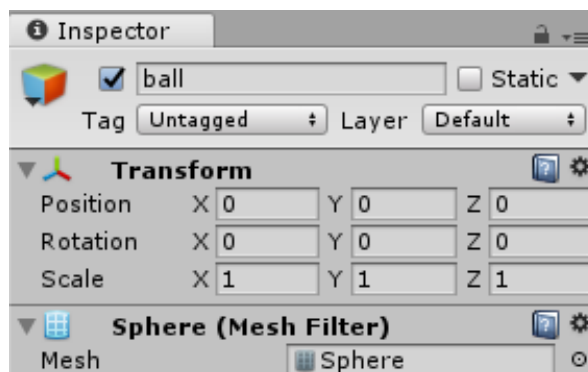
7. Cambia el nombre y coloca los valores como:



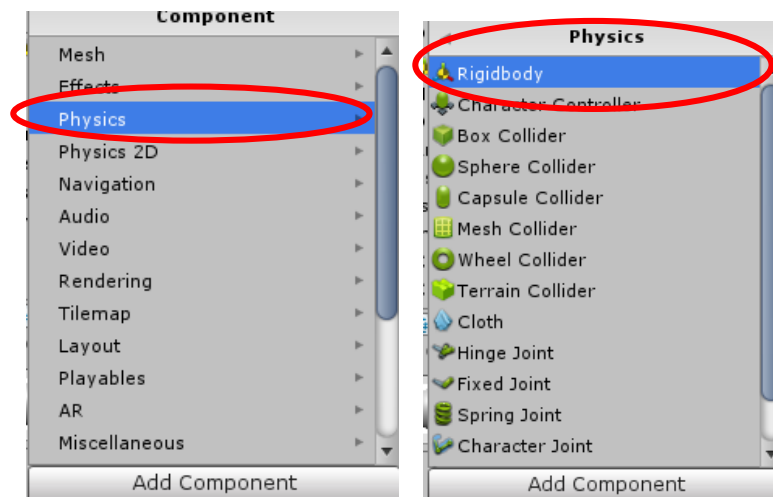
8. La vista del juego debe tener la siguiente apariencia:



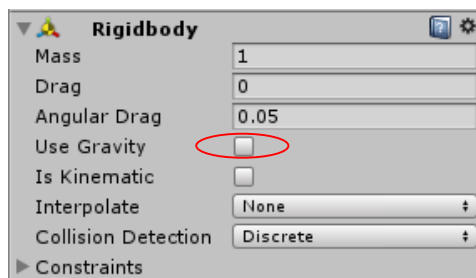
9. Ahora agrega un objeto 3D sphere y cambia sus propiedades a:



10. Agrega un componente Rigidbody, esto permitirá que el objeto ball pueda comportarse como un elemento rígido y por lo tanto rebotar:



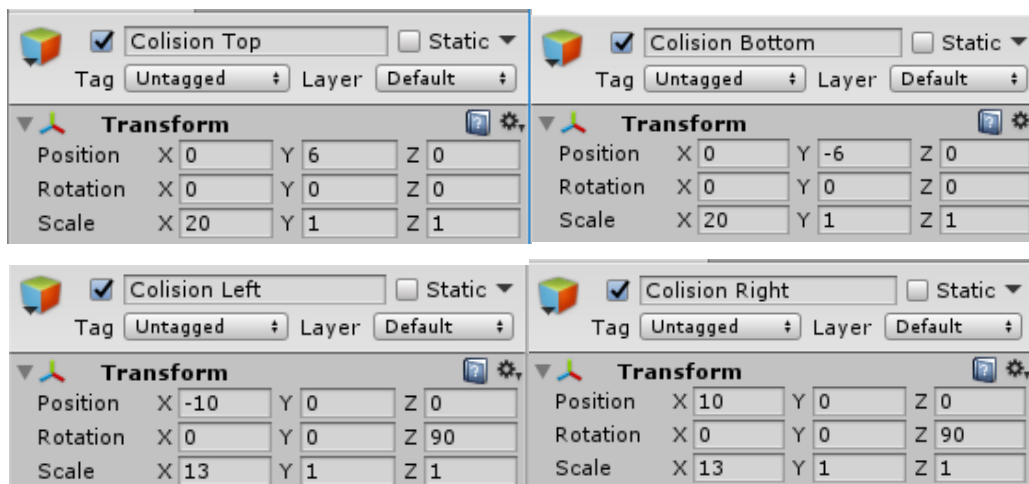
11. Cambia los atributos del Rigidbody, no olvides desactivar la gravedad:



12. Ahora agregaremos los bordes donde rebotará la bola. Para ello agregar 4 objetos cubo con los identificadores:

- Colision Top
- Colision Bottom
- ColisionLeft
- Colision Right

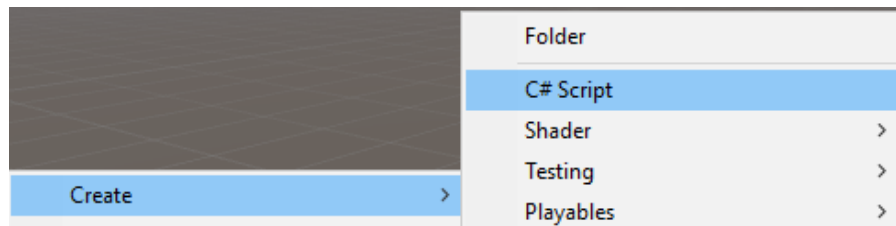
Establece en cada uno de ellos las siguientes propiedades:



La disposición final de los elementos en la ventana Game, se debe apreciar como se muestra:



13. Crear el script que permita mover la bola, para ello efectúe clic derecho sobre la ventana de assets y elija la opción Create, luego C# Script y nombre al script como Ball.



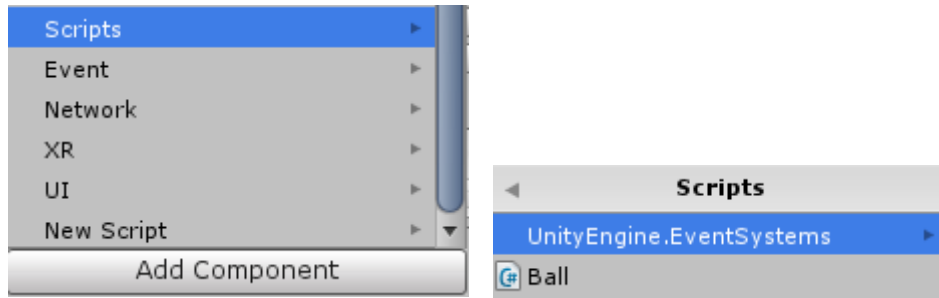
Agregue el siguiente código al script Ball

```

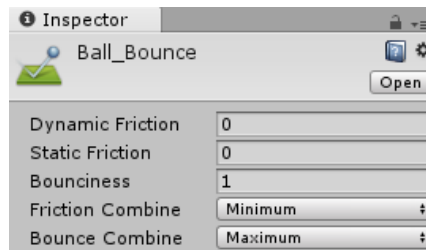
bola.cs* [icon] [X]
Assembly-CSharp [icon] bola
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class bola : MonoBehaviour
6  {
7
8      public float velocidad;
9      private Vector3 direccion;
10     // Use this for initialization
11     void Start()
12     {
13         this.direccion = new Vector3(1f, 1f, 0f);
14     }
15
16     // Update is called once per frame
17     void Update()
18     {
19         this.transform.position += direccion * velocidad;
20         transform.Rotate(0, 5, 0);
21     }
22     private void OnCollisionEnter(Collision col)
23     {
24         Vector3 normal = col.contacts[0].normal;
25         direccion = Vector3.Reflect(direccion, normal);
26     }
27 }
28

```

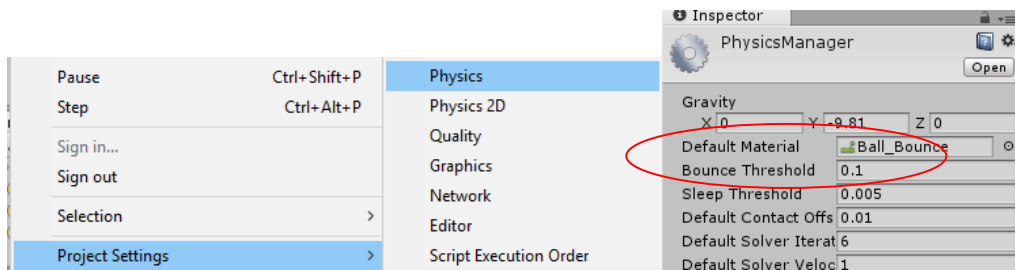
14. Establezca el script Ball para que sea ejecutado por la sphere, para ello en la ventana del inspector elija la opción Add Component → Scripts → y elija Ball. También puede arrastrar el script Ball hacia la sphere.



15. Para lograr el rebote crearemos un nuevo material de físicas, al que llamaremos Ball_bounce, establezca las siguientes propiedades:



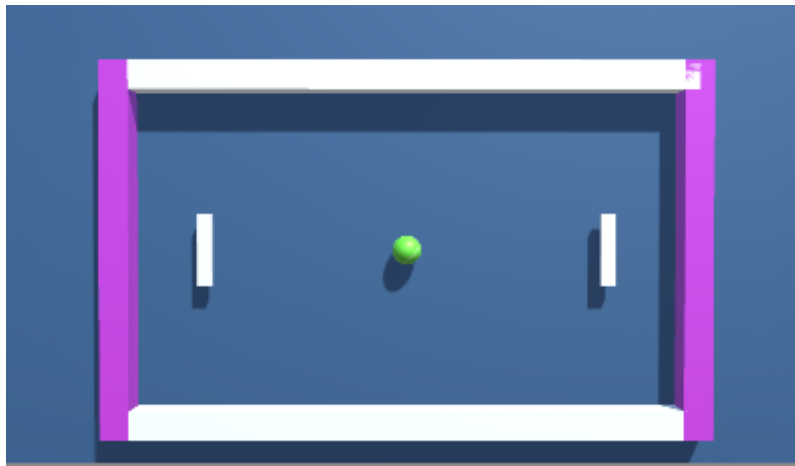
Establezca la física a usar por todo el proyecto, para ello en el menú Edit → Project Settings, elija Physics, luego en el inspector establezca el default material como Ball_Bounce, además de la propiedad Bounce threshold en la mínima posible, asigne el valor 0.1



16. Ejecute el juego haciendo clic en el botón de control play:



Modifique el color de las paredes izquierda y derecha, ya que actuaran como TARGETS.



Modificar el script bola.cs

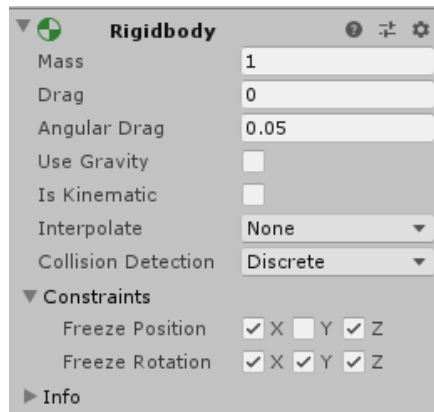
```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  public class bola : MonoBehaviour
5  {
6      public float velocidad;
7      private Vector3 direccion;
8      public Vector3 Punto_regeneracion;
9      void Start()
10     {
11         this.direccion = new Vector3(1f, 1f, 0f);
12     }
13     void Update()
14     {
15         this.transform.position += direccion * velocidad;
16         transform.Rotate(0, 5, 0);
17     }
18     private void OnCollisionEnter(Collision col)
19     {
20         Vector3 normal = col.contacts[0].normal;
21         direccion = Vector3.Reflect(direccion, normal);
22         if (col.gameObject.name=="ColisionLeft")
23         {
24             transform.position = Punto_regeneracion;
25         }
26         if (col.gameObject.name == "ColisionRight")
27         {
28             transform.position = Punto_regeneracion;
29         }
30     }

```

AGREGANDO MOVIMIENTO A LAS PALETAS (Player1 y Player2)

Agregue un objeto rigidbody para cada paleta:



Agregar dos scripts para cada paleta

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Player1 : MonoBehaviour
6  {
7      public float velocidad=3;
8      public Rigidbody rb;
9      // Use this for initialization
10     void Start()
11     {
12         rb = GetComponent<Rigidbody>();
13     }
14
15     // Update is called once per frame
16     void Update()
17     {
18         if (Input.GetKey(KeyCode.W))
19         {
20             rb.velocity = new Vector3(0, velocidad, 0 );
21         }
22         else if (Input.GetKey(KeyCode.S))
23         {
24             rb.velocity = new Vector3(0, -velocidad, 0 );
25         }
26         else
27         {
28             rb.velocity = new Vector3(0, 0, 0);
29         }
30     }
31 }
32
33

```



```

Player2.cs*  Player1.cs*  bola.cs
Assembly-CSharp  Player2
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Player2 : MonoBehaviour
6  {
7      public float velocidad = 3;
8      public Rigidbody rb;
9      // Use this for initialization
10     void Start()
11     {
12         rb = GetComponent<Rigidbody>();
13     }
14
15     // Update is called once per frame
16     void Update()
17     {
18         if (Input.GetKey(KeyCode.UpArrow))
19         {
20             rb.velocity = new Vector3(0, velocidad, 0);
21         }
22         else if (Input.GetKey(KeyCode.DownArrow))
23         {
24             rb.velocity = new Vector3(0, -velocidad, 0);
25         }
26         else
27         {
28             rb.velocity = new Vector3(0, 0, 0);
29         }
30     }
31 }
32
33

```

AGREGANDO UN CONTADOR DE PUNTAJE

Agregar en el script bola.cs: la referencia a la librería: UnityEngine.UI;

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class bola : MonoBehaviour

```

Agregar las siguientes propiedades a la clase bola:

```

[SerializeField]
private int player1puntos;
[SerializeField]
private int player2puntos;

public Text player1Text;
public Text player2Text;

```

Inicializar los valores en el método Start:

```

void Start()
{
    this.direccion = new Vector3(1f, 1f, 0f);
    player1puntos = 0;
    player2puntos = 0;
}

```

Agregar la actualización en el update:

```

void Update()
{
    this.transform.position += direccion * velocidad;
    transform.Rotate(0, 5, 0);
    player1Text.text = player1puntos.ToString();
    player2Text.text = player2puntos.ToString();
}

```

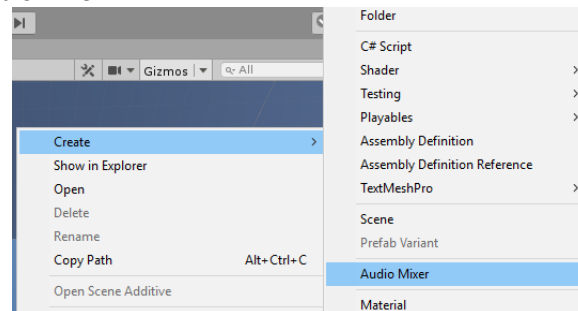
Modificar el método OnCollisionEnter

```
private void OnCollisionEnter(Collision col)
{
    Vector3 normal = col.contacts[0].normal;
    direccion = Vector3.Reflect(direccion, normal);
    if (col.gameObject.name=="ColisionLeft")
    {
        transform.position = Punto_regeneracion;
        player1puntos++;
    }
    if (col.gameObject.name == "ColisionRight")
    {
        transform.position = Punto_regeneracion;
        player2puntos++;
    }
}
```

AGREGAR LOS EFECTOS DE SONIDO

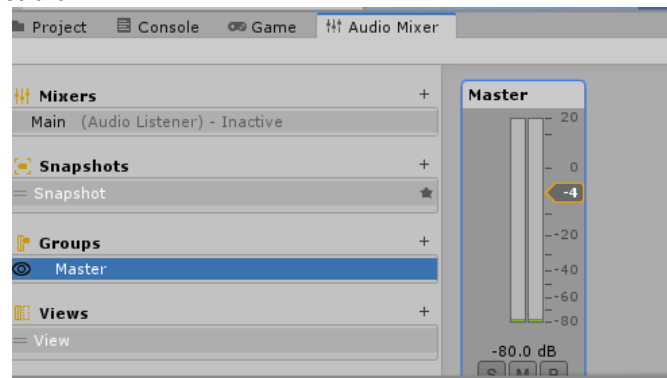
Crear una carpeta denominada Audio en el proyecto e importar los assets necesarios.

Crear un GameObject audio mixer:

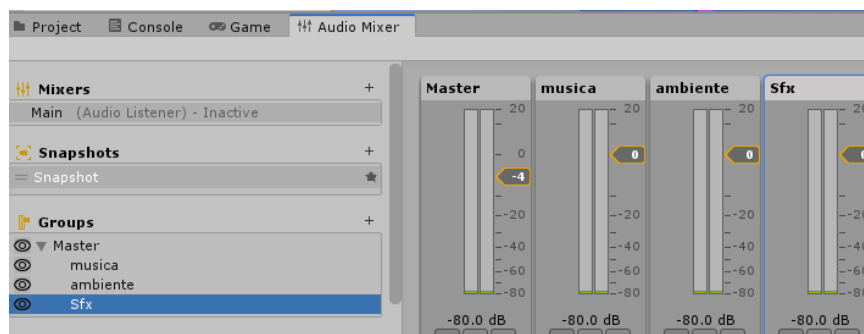


Nombralo como Main

Al seleccionar se mostrará :



Seleccione groups y su botón + para agregar los grupos música, ambiente y Sfx, quedará de la siguiente manera:

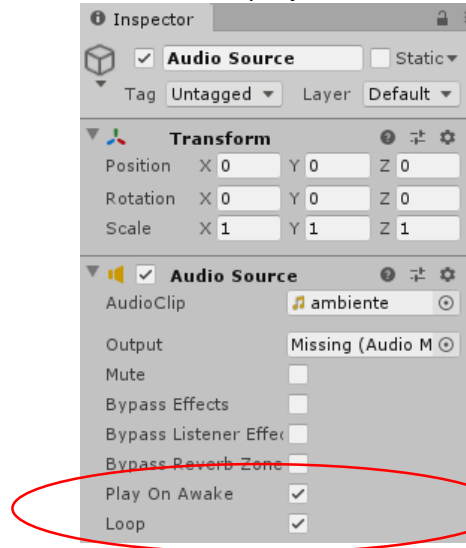


Este será el control de audio de todo el juego.

Agregar GameObjects y estructurarlos de modo que se obtenga la estructura:



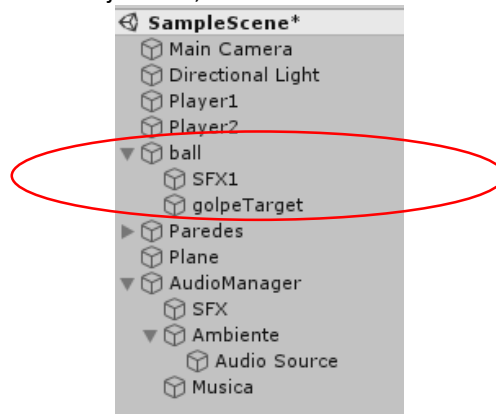
Por ejemplo Agregue un AudioSource a ambiente y elija el archivo ambiente.



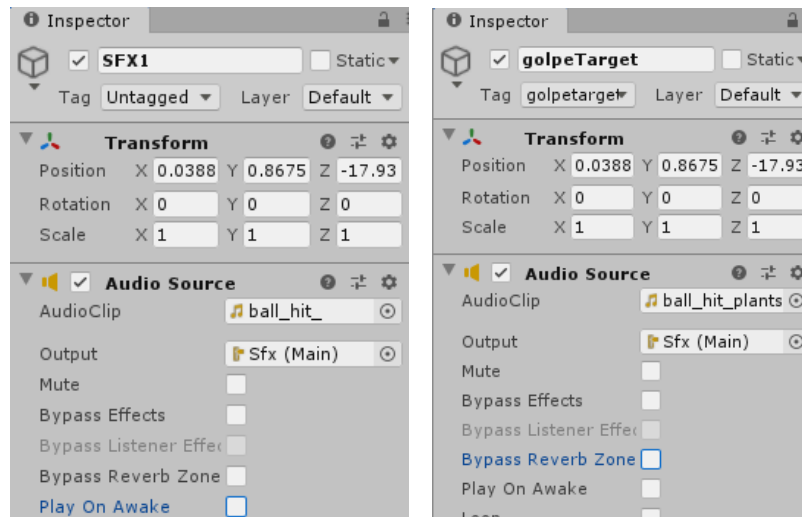
Con esto tendremos música ambiental que se inicia con el juego y se repite constantemente al culminar.

AGREGAR EL EFECTO DE CHOQUE CON PAREDES Y PLAYERS.

Insertar dos objetos audiosource al objeto ball, como se muestra:



Configurar sus orígenes:



Modificar en el script de ball:

```

Player2.cs Player1.cs bola.cs* X
Assembly-CSharp
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.UI;
5 public class bola : MonoBehaviour
6 {
7     public float velocidad = 1;
8     private Vector3 direccion;
9     public Vector3 Punto_regeneracion;
10
11     public int player1puntos;
12     public int player2puntos;
13
14     public Text player1Text;
15     public Text player2Text;
16
17     AudioSource _source;
18     GameObject clipAudio;
19     public void Awake()
20     {
21         _source = GetComponentInChildren<AudioSource>();
22         clipAudio = GameObject.FindGameObjectWithTag("golpetarget");
23     }
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

private void OnCollisionEnter(Collision col)
{
    Vector3 normal = col.contacts[0].normal;
    direccion = Vector3.Reflect(direccion, normal);
    if (col.gameObject.name=="CollisionLeft")
    {
        transform.position = Punto_regeneracion;
        player1puntos++;
        clipAudio.GetComponent<AudioSource>().Play();
    }
    if (col.gameObject.name == "CollisionRight")
    {
        transform.position = Punto_regeneracion;
        player2puntos++;
        clipAudio.GetComponent<AudioSource>().Play();
    }
    if (_source)
        _source.Play();
}

```

Glosario de la Unidad 1

Asset: Un asset es una representación de cualquier item que puede ser utilizado en su juego o proyecto. Un asset podría venir de un archivo creado afuera de Unity, tal como un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos que Unity soporta. (Unity Technologies, 2018)

Game Object: Representan personajes, propiedades y el escenario para ello deben agrupar o contener objetos denominados Components integrados por personajes, luz, Cubos, sonidos, imágenes y otros. (Unity Technologies, 2018)

Clase: Definición de las características y comportamiento de uno o muchos objetos.

Componente transformar - Transform Component: es uno de los más importantes Components, habilita todas las propiedades Transform del GameObject, por lo que define la posición, rotación, y escala del GameObject en la Escena (Unity Technologies, 2018)

Colliders: definen la forma de un objeto para los propósitos de colisiones físicas, un collider, es invisible, necesita no estar con la misma forma exacta que el mesh del objeto (Unity Technologies, 2018)

Escena – Scene: contienen los objetos del juego. Pueden ser usadas para crear un menú principal, niveles individuales, y cualquier otra cosa. Piense en cada archivo de escena, como un nivel único. (Unity Technologies, 2018)

Materials son definiciones acerca de cómo la superficie debería ser renderizada, incluyendo referencias a texturas utilizadas, información del tiling (suelo de baldosas), tines de color y más. Las opciones disponibles para un material depende en qué shader del materia está utilizando (Unity Technologies, 2018)

Objeto: Es el código funcional instanciado desde una clase, por lo que obtiene las definiciones de características y comportamiento.

Rigidbody: Un Rigidbody es el componente principal que permite el comportamiento físico para un objeto. Con un Rigidbody adjunto, el objeto inmediatamente responderá a la gravedad. Si uno o más componentes Collider son también agregados entonces el objeto será movido por colisiones entrantes. (Unity Technologies, 2018)

(Incorpore 8 términos como mínimo ordenados alfabéticamente, la definición considerada debe de estar correctamente referenciada con una cita en estilo APA)



Bibliografía de la Unidad 1

(Sin numeración, ordenado alfabéticamente y en formato de presentación APA)

- Adrigm. (30 de 04 de 2018). *zehngames*. Obtenido de <http://www.zehngames.com/comunidad/tema/tecnicas-e-ideales-de-diseno-de-videojuegos/>
- Roy. (30 de 04 de 2018). *otakufreaks*. Obtenido de <http://www.otakufreaks.com/historia-de-los-videojuegos-el-origen-y-los-inicios/>
- Unity Technologies. (30 de 4 de 2018). *Manual de Unity 3D*. Obtenido de <https://docs.unity3d.com/es>



Universidad
Continental



HUANCAYO

Av. San Carlos 1980
Urb. San Antonio - Huancayo

Teléfono: 064 481430

LOS OLIVOS - LIMA

Av. Alfredo Mendiola 5210
Los Olivos - Lima

Teléfono: 01 2132760

MIRAFLORES - LIMA

Jr. Junín 355
Miraflores - Lima

Teléfono: 01 2132760

AREQUIPA

Av. Los Incas s/n
Urb. Lambramani, José Luis
Bustamante y Rivero - Arequipa

Teléfono: 054 412030

CUSCO

Av. Collasuyo Lote B-13
Urb. Manuel Prado
Wanchaq - Cusco

Teléfono: 084 480070

ucontinental.edu.pe

