ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

INTEGRACIÓN DE UN MOTOR 3D CON UNA HERRAMIENTA DE MODELADO Y ANIMACIÓN PARA CREAR MUNDOS VIRTUALES INTERACTIVOS

TESIS DE GRADO

PREVIO LA OBTENCIÓN DEL TÍTULO DE:
INGENIERO EN SISTEMAS INFORMÁTICOS

AUTORES:

Pablo Antonio Romero Santillán
Ximena Valeria Velasco García

RIOBAMBA – ECUADOR
2014

AGRADECIMIENTO

A la Escuela Superior Politécnica de Chimborazo y especialmente a la Facultad de Informática y Electrónica por ser quienes nos han formado en estos años.

A nuestras familias que son el pilar fundamental de nuestras vidas.

DEDICATORIA

A nuestras familias.

FIRMAS RESPONSABLES Y NOTAS

	FIRMA	**FECHA**
Ing. Iván Menes **DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRÓNICA**	_____	_____
Ing. Jorge Huilca **DIRECTOR DE LA ESCUELA DE INGENIERÍA EN SISTEMAS**	_____	_____
Ing. Fernando Proaño **DIRECTOR DE TESIS**	_____	_____
Dr. Alonso Álvarez **MIEMBRO DEL TRIBUNAL DE TESIS**	_____	_____
DIRECTOR DEL CENTRO DE DOCUMENTACIÓN	_____	_____

AUTORÍA

"Nosotros Ximena Valeria Velasco García y Pablo Antonio Romero Santillán, somos los responsables de las ideas, doctrinas y resultados expuestos en esta Tesis, y el patrimonio intelectual de la misma pertenecen a la Escuela Superior Politécnica de Chimborazo".

Ximena Valeria Velasco García

Pablo Antonio Romero Santillán

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

INTRODUCCIÓN

Por medio del presente proyecto de tesis se pretende la creación de una Metodología para integrar una herramienta de animación y modelado con un motor 3D para la creación de mundos virtuales, en este caso un mundo virtual de la Escuela Superior Politécnica de Chimborazo (ESPOCH). La metodología permitirá importar los archivos necesarios desde la herramienta de modelado y animación hacia el motor 3D para posteriormente ser programados lo que generará como resultado una aplicación que será publicada en la página de la ESPOCH http://www.espoch.edu.ec, para ser accesible desde la Web.

Previa a la realización de este trabajo investigativo se realizó un análisis de la necesidad de un Campus Virtual de la ESPOCH por medio de una encuesta que emitió como resultados que un 100% de personas entre 15 y 25 años creen que será útil la creación de un mundo virtual para conocer la ESPOCH, sin la necesidad de viajar para ahorrar tiempo y para tener un conocimiento previo del Campus cuando deseen viajar a visitarlo.

El presente documento se encuentra dividido en 5 capítulos. El primer capítulo es el Marco Propositivo en el cual se dará el punto de partida para la investigación. El segundo capítulo muestra las generalidades de los mundos virtuales y conceptos básicos relacionados a ellos; el tercer capítulo habla sobre los conceptos básicos y la terminología utilizada en la metodología y en las herramientas que se utilizarán para la creación de la aplicación. En el cuarto capítulo se presenta el proyecto de trabajo que será utilizado para la metodología y se realizará una descripción y comparación de las herramientas de animación y modelado y motores 3D más conocidos en el medio, para proceder a una posterior selección del software más adecuado. En el capítulo final se implementa la metodología creada y se describe paso a paso como se realiza la exportación, importación y programación de los elementos necesarios para la creación del mundo virtual de la ESPOCH.

CAPITULO I

MARCO PROPOSITIVO

1.1. Antecedentes

Los motores 3D han sido considerados herramientas fundamentales en los diseños de campos virtuales, pero presenta algunos problemas en el momento de su aplicación, entre los que se puede mencionar los siguientes:

a. Los motores 3D fueron creados para generar programas, de acuerdo a una necesidad preestablecida y por sí mismos no están hechos para modelado de imágenes en 3D.
b. Para aplicaciones en campos virtuales los motores 3D, deben complementarse con programas de modelado, que permitan la transferencia de la información necesaria.
c. Pueden existir varias alternativas de motores 3D, pero no cuentan con la configuración necesaria para adaptarse a los programas de modelado, por esta razón se hace indispensable la selección óptima de los motores 3D que no entorpezcan lo que se quiere alcanzar en el diseño de un campus virtual

d. Lo mismo ocurre con los programas de modelado, es importante una selección correcta de la herramienta apropiada para que sea compatible con los motores 3D.
e. La aplicación virtual debe funcionar con un rendimiento óptimo en la mayoría de navegadores Web, para lograr una mayor participación de cualquier usuario.

Todos estos aspectos mencionados, pueden provocar inconvenientes durante el proceso de aplicabilidad, haciendo que el resultado final no sea el esperado y además no se estaría aprovechando el potencial tanto del programador como del modelador.

Para resolver el problema planteado, se propondrá el uso de un motor 3D complementado con un programa de modelado y animación, como alternativas en el diseño de campos virtuales interactivos.

1.2. Justificación

El tema elegido para este trabajo de investigación obedece a una necesidad vista por medio de la tabulación de datos de una encuesta (Anexo 1) que se presentó a posibles usuarios de un campus virtual de la ESPOCH por estar cursando, o cerca de cursar, sus estudios universitarios; todos ellos afirman que la creación de un Mundo Virtual permitirá un conocimiento previo y rápido de la Institución sin necesidad de encontrase en ella físicamente. Para conseguir este objetivo se pretenderá encontrar alternativas factibles para el diseño de Entornos Virtuales, basados en el uso de motores 3D y herramientas de modelado y animación 3D, que sean accesibles e interactivos desde la Web.

Para lograr y demostrar su aplicación en el diseño de Campus Virtuales, este proyecto pretende optimizar las bondades que ofrecen los motores 3D libres como son su bajo costo comparándolo con motores licenciados, amigabilidad de la interfaz, sencillez en el manejo por parte del usuario, facilidad de exportación para múltiples plataformas incluidas plataformas Web.

De la misma manera se analizarán las bondades ofrecidas por programas de modelado gratuito entre las cuales se analizaran la comunidad de soporte para el desarrollo, la competitividad entre programas similares de modelación y animación, entre otros parámetros.

Esta investigación beneficia principalmente a los miembros de la comunidad Politécnica (docentes, estudiantes, empleados y trabajadores). Además beneficiará a personas que necesiten información sobre la ESPOCH, prácticamente sin importar el lugar donde se encuentren, con el requerimiento de un buen acceso a la Web.

1.2.1. Justificación Práctica

La utilidad práctica se manifiesta en la posibilidad de aplicar este diseño en la misma ESPOCH, que actualmente no cuenta con un campus virtual interactivo y accesible desde la Web.
El mundo virtual a presentarse constará de los edificios más significativos de la ESPOCH y de la infraestructura interna deledificio de la Facultad de Informática y Electrónica.

1.2.2. Justificación Teórica

La investigación permitirá la posibilidad de replicar el estudio en otras universidades con características similares, y ser un referente para posibles comparaciones de resultados (mejores prácticas) a nivel del país.
La utilidad teórica del presente trabajo se manifiesta en que sus resultados pueden servir de base para futuros estudios ampliatorios y en el aporte documentado que se presenta del tema tratado.

Adicionalmente esta aplicación permitirá utilizar las técnicas con los últimos adelantos en este campo y permitirá situar a la ESPOCH en un nivel competitivo, más aun hoy, que existe un proceso de acreditación a nivel nacional.

1.3. Objetivos

1.3.1. Objetivo General

Generar una metodología para integrar un Motor 3D con una herramienta de Modelado y Animación para crear un mundo virtual de la Escuela Superior Politécnica de Chimborazo.

1.3.2. Objetivos Específicos

- Fundamentar científicamente la compatibilidad de motores 3D con programas de modelado para determinar que herramientas de modelado son las óptimas para diseñar componentes visuales en 3D.
- Proponer una guía metodológica para integrar un motor 3D con una herramienta de modelado y animación.
- Importar el componente visual creado en una herramienta de modelado en 3D hacia el motor de video juegos.
- Programar los componentes necesarios para la interacción del usuario dentro del mundo virtual, utilizando un motor 3D.
- Desarrollar un campus virtual interactivo para la ESPOCH, el cual será accesible desde la Web.
- Evaluar el correcto funcionamiento de la aplicación generada.

1.4. HIPÓTESIS

La implementación de una metodología permitirá lograr la correcta integración de una herramienta de modelado y animación con un motor 3D y medir el rendimiento de la aplicación del Campus Virtual.

CAPITULO II

GENERALIDADES DE LOS MUNDOS VIRTUALES

2.1. Realidad Virtual

"Realidad virtual: un sistema de computación usado para crear un mundo artificial en el cual el usuario tiene la impresión de estar y la habilidad de navegar y manipular objetos en él" [1].

Para el presente documento se definirá a la realidad virtual como una ciencia que por medio del empleo de computadores y otros dispositivos, permiten la creación y generación de aplicaciones en tiempo real, que utilizando conjunto de imágenes, sonidos, etc, dan al usuario la sensación de encontrarse en el lugar representado y dependiendo del nivel de inmersión le permite interactuar con el mundo virtual.

2.1.1. Las 3 “I” de la Realidad Virtual

La realidad virtual cuenta con tres elementos principales como se detalla en la Figura 1, los cuales se describen a continuación:

Inmersión: Permite al usuario “desconectarse” del mundo real y solo percibir estímulos del mundo virtual sintiéndose parte de él.
Interacción: El usuario debe tener control total del mundo y sus elementos, debe ser capaz de moverse, navegar, manipular e interactuar con objetos, todo esto a través de dispositivos externos.
Imaginación: Permite al usuario desarrollar la capacidad de percibir cosas que no existen.

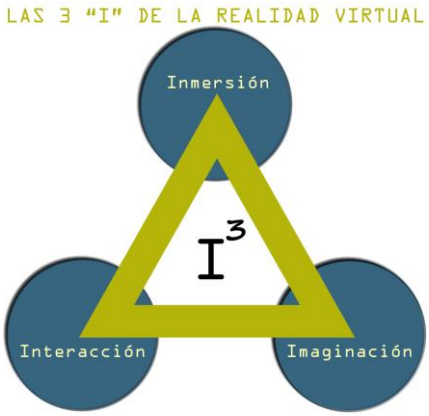


Figura 1. Elementos dela Realidad Virtual

(Fuente: Carmen Ortigueira España et al [Online] http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/Realidad%20Virtual/web/introduccion.html)

2.1.2.Características de la Realidad Virtual

Presencia: Por medio del uso de cualquier dispositivo de entrada debe existir la presencia del usuario dentro del entorno.
Interrelación: Por medios de la realidad virtual es posible la interacción interpersonal a pesar de no encontrarse en el mismo lugar físico o al mismo tiempo.

Facilidad de navegación: la realidad virtual debe permitir sobre todo la facilidad para cambiar el punto de observación del mundo de acuerdo a sus movimientos.
Similitud con el mundo real: La realidad virtual guarda cierto grado de similitud con el mundo físico real y captura los movimientos y acciones naturales del usuario y los proyecta en el mundo virtual, permitiendo así una experiencia tan real como encontrarse personalmente dentro de éste.
Libertad de creación: Permite la libre creación, ya que generalmente se pretende simular un mundo alternativo en el cual no existen límites al momento del desarrollo.

2.1.3. Clasificación de los Sistemas de Realidad Virtual

Sistemas inmersivos que se utilizan en realidad virtual están ligados al uso de dispositivos externos como cascos, guantes, chalecos, HMD, entre otros que se encargan de capturar el movimiento del cuerpo o de sus partes (posición y rotación) y transmitirlos a un ambiente tridimensional previamente generado usando un ordenador, permitiéndolos interactuar y desplazarse en él.

Sistemas no inmersivos también conocidos como sistemas de ventanas, o realidad virtual de escritorio no precisa de dispositivos externos adicionales, pero de la misma manera permite al usuario interactuar en tiempo real con ambientes, espacios, personajes (controlados por el ordenador o por otros usuarios), todo esto en una ventana de escritorio, valiéndose de otro tipo de medios como por ejemplo Internet. Ej: Paseos virtuales, video juegos.

Cabinas de simulación: Permiten al usuario la experiencia de un ambiente simulado dentro de una cabina. Ej: simuladores de vuelo.

Realidad Aumentada: Permite al usuario “complementar” el mundo real a través de imágenes proyectadas siempre valiéndose del uso de dispositivos adicionales.

Sistemas de telepresencia: Son usados para realizar tareas a distancia, se operan remotamente con un dispositivo especial manejado por el usuario y simuladas por un agente.

2.2. Interactividad

"Interactividad es la capacidad del receptor para controlar un mensaje no-lineal hasta el grado establecido por el emisor, dentro de los límites del medio de comunicación asincrónico" [2].

Para entender este concepto de interactividad se tendrán en cuenta ciertos aspectos:

Todo medio de comunicación cuenta con cuatro elementos: emisor, mensaje, medio, receptor como se puede ver la Figura 2.

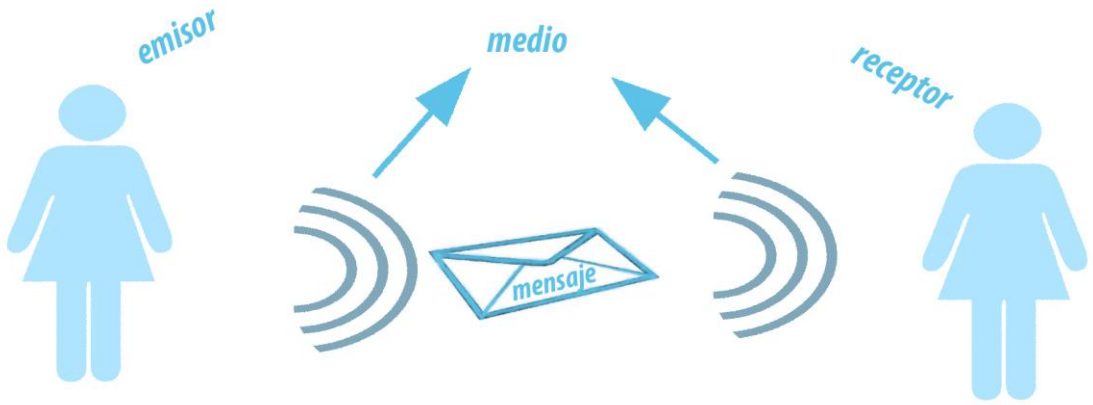


Figura 2. Elementos de un medio de comunicación
(Fuente: Propia)

El medio puede ser no interactivo como la televisión, o interactivo como una página Web que contiene enlaces a otras páginas.

Otra característica de los medios es la sincronía. Los medios sincrónicos solo transmiten información si el emisor y el receptor se han puesto de acuerdo para hacerlo, existen también los medios asíncronos los cuales transmiten información sin necesidad de un previo acuerdo.

El mensaje puede ser lineal como un libro al que se lo lee una página después de otra, o no lineal como un CD –ROM que tiene varios archivos y puede abrirse el que se desee.
Solo se puede dar en medios de comunicación no-lineales y asíncronos.
La interactividad permite que el emisor al recibir un estímulo genere y envié una respuesta.

2.3. Gráficos 3D

La creación de gráficos 3D se lleva a cabo por un proceso de cálculos matemáticos realizados sobre entidades geométricas 3D originadas con ayuda de un computadory programas especiales de manejo 3D; éstas imágenes, sin dejar de ser bidimensionales, tiene como función conseguir que el usuario perciba la sensación de 3D al ser mostradas en una pantalla o estar impresas en papel.
El proceso de creación de gráficos 3D, como se muestra en la Figura 3, por computadora puede ser dividido en estas tres fases básicas:

- Modelado
- Composición de la escena
- Renderizado (creación de la imagen final)

Figura 3. Creación de Gráficos 3D
(Fuente: Propia)

2.3.1. Modelado

El modelado es una etapa en la que se crean de manera individual todos los objetos que van a formar parte de la escena.

El modelado 3D permite la representación de un objeto del mundo real en el computador a través de un conjunto de elementos, propiedades y características que van a ser procesados y al ser vistos en un monitor darán la impresión al usuario de un objeto real en 3 dimensiones.

Existen diversas técnicas de modelado entre las cuales pueden citarse las siguientes: Constructive Solid Geometry, modelado con B-splines racionales no uniformes (NURBS), Image Based Modeling (IBM), modelado poligonal o subdivisión de superficies.

2.3.2. Composición de la Escena

Esta etapa incluye la distribución de modelos, iluminación, cámaras y demás elementos, el proceso de tessalation y descomposición de los objetos en mallas, para continuar con la animación de los mismos.

La iluminación constituye un factor directo en la escena ya que de eta dependerá el resultado estético y la calidad visual del trabajo finalizado, tomando en cuenta toda la iluminación que podría recibir un objeto en la vida real, de acuerdo a las fuentes de luz que se incluyan en la escena.

El siguiente proceso, que es un proceso previo a la renderización, consiste en la transformación de los objetos desde un punto en una esfera hacia la representación poligonal de una esfera, este proceso es conocido como tesselation. En este paso se procede a la descomposición de los objetos de representaciones primitivas (esferas, conos, etc.), hacia redes de triángulos interconectados conocidos como mallas.

La animación es el proceso que pretende imitar la realidad mediante el uso de esqueletos, deformadores, dinámicas, o transformaciones en los tres ejes, que ayuden al objeto a cambiar su forma, ubicación, características, etc.

2.3.3. Renderización

La renderización es el proceso de generar imágenes 2D o animaciones a partir de la escena creada. Este proceso es comparado a capturar en un video o una fotografía la escena terminada.

Este proceso necesita de una gran capacidad de cálculo debido a que se requiere simular una gran cantidad de procesos físicos complejos.

2.4. Mundos Virtuales

Un mundo virtual es un ambiente tridimensional, persistente, simulado por medio de un computador, que busca imitar y da la impresión de un mundo real, aunque no sea necesariamente un lugar físico existente, que se encuentra disponible para un gran número de usuarios 24 horas al día los 7 días de la semana y no deja de existir aunque los usuarios se encuentren desconectados. Estos usuarios interactúan unos con otros o con el ambiente en tiempo real por medio de Avatares que son representaciones, alter-egos o personajes en 3 dimensiones creados por ellos mismos para ser su representación en el mundo virtual.

Los mundos virtuales en la actualidad están constituidos principalmente por los metaversos, y MMORPG`s estas siglas en inglés significan “Massively Multiplayer Online-Role Playing Game”, en su traducción literaria “juegos masivos de rol de múltiples jugadores en línea”.

2.4.1. Características de los mundos virtuales

1. Inmersión:

   La inmersión está ligada a dos factores importantes, la conexión con los sentidos y la psicología que implica.

   Para percibir el mundo real utilizamos nuestros sentidos, por lo tanto un mundo virtual debe ser percibido de la misma manera, principalmente son usados los sentidos de la vista y el oído para percibir estos mundos sin la necesidad de utilizar dispositivos de entrada/salida adicionales.

   A mayor nivel de inmersión mejor experiencia en el mundo virtual.

2. Tridimensionalidad

   Un mundo virtual debe permitir que el usuario se sienta realmente en él, sin importar las limitaciones de hardware con las que se cuenten. Los modelos en tres dimensiones permiten acercarse más a la realidad, pero se debe buscar un equilibrio, llegar al punto en el que la realidad virtual no sea tan real de manera que sea manejable por el cerebro humano.

3. Interfaces para Avatares

   Un Avatar al ser la representación en tres dimensiones de un usuario en un mundo virtual, debe permitirle a éste personalizarlo en varios aspectos, de tal manera que si así lo quiere pueda guardar cierta similitud con el usuario en la vida real.

   Un Avatar puede realizar al menos las acciones básicas de un humano como caminar, correr, conversar, moverse, etc.

   El Avatar es controlado mediante el teclado y el mouse, éste se mueve alrededor del mundo para interactuar con él y con otros avatares.

4. Basada en internet

   Se debe tomar en cuenta que un mundo virtual, al igual que cualquier página Web, se encuentra alojado en un servidor al cual se accederá desde un navegador Web. Esto permite que el mundo siempre se encuentre disponible.

5. Persistencia

   Un mundo virtual es persistente lo que significa que está disponible 24 horas al día, 7 días a la semana, por lo tanto un usuario puede logearse y seguir presenciando lo que está sucediendo, mientras que cuando se desconecta el resto de usuarios seguirán con sus actividades dentro del mundo.

6. Interactividad

   En un mundo virtual nunca se está solo, a menos que el usuario así lo decida, mediante el Avatar todos los usuarios pueden interactuar manteniendo su privacidad si así lo quieren.

7. Sincronía

   De acuerdo a ciertas teorías de la física el ser humano existe en cuatro dimensiones, 3 en el espacio y 1 en el tiempo. Mediante la sincronía se pueden realizar actividades y eventos al mismo tiempo con otros usuarios tal como se los realizaría en la vida real.

CAPITULO III

LOS MOTORES PARA MUNDOS VIRTUALES

3.1. Motor Genérico (Engine)

Un motor es un tipo de software que genera código y produce elementos que dan inicio a ciertos procesos automatizados, varios de los elementos de este software trabajan de tal manera que se minimice la intervención humana y permiten el mantenimiento del software generado en tiempo real.
Un motor es usado como la base para construir un juego debido a que permite que las tareas repetitivas o genéricas sean realizadas de una manera más rápida para que el programador pueda dedicarse a otro tipo de trabajos no genéricos.

3.2. Motor 3D (Motor de Juegos)

Se usa el término motor 3D o motor de juego al referirse a un software que permite la integración de elementos 2D y 3D para la posterior programación, desarrollo y creación de juegos de video y mundos virtuales para computadora, consolas de juego, dispositivos móviles, etc. El motor 3D busca principalmente proveer al videojuego de un motor de renderizado para los gráficos, motor físico o detector de colisiones, sonidos, scripting, animación, inteligencia artificial, administración de memoria y un escenario gráfico.

3.3. Assets

En lenguaje informático, específicamente en el diseño 3D, un asset es un objeto que se encuentran dentro del juego, estos elementos deben ser codificados para que funcionen.
Entre los assets más importantes están los modelos, animaciones, sonidos, físicas, etc., que son los elementos más importantes dentro de un juego.

3.4. Interfaz de Programación de Aplicaciones (*API*)

Es un sistema de rutinas, de protocolos y de herramientas para desarrollar aplicaciones de software.
Una buena API se encarga de especificar como cada componente de un software interactúa con los otros, y hace más fácil desarrollar un programa proporcionando todos los bloques del desarrollo del programa. El programador pone los bloques juntos.
Entre estos los más importantes son el DirectX (de Microsoft) y el OpenGL (que trabaja con la mayoría de los sistemas operativos).

3.5. Espacio Tridimensional

El espacio tridimensional es un espacio virtual matemático que es creado por un programa de diseño 3D.
Este espacio cuenta con un punto conocido como origen de donde nacen las líneas virtuales, y se encuentra definido en un sistema cartesiano de tres ejes: X,Y,Z.
Dentro de este espacio se encontraran todos los objetos tridimensionales que van a componer la escena.

3.6. Objetos 3D

Los objetos se almacenan por puntos en el espacio tridimensional, llamados vértices. Los vértices van formando polígonos; cuanto más polígonos posea un objeto, más complicado se hace, lo que quiere decir que lleva más tiempo de procesamiento pero es más detallado. El mundo virtual no necesita saber cuántos objetos hay en memoria o cómo van a ser mostrados, sólo le interesa que sean desplegados de la forma correcta, y que el modelo esté en el cuadro correcto de la animación.

3.7. Culling

Culling es una técnica que remueve elementos que no deben ser mostrados en la imagen final de una escena debido a que otros elementos que forman parte de la misma los obstaculizan y no permiten su visualización, o simplemente porque se encuentran fuera del ángulo de vista como se puede apreciar en la Figura 4. Mediante esta técnica se logra que no se tome tiempo innecesario de renderizado, así se reduce la cantidad de trabajo del motor. El Culling es más fácil de implementar en juegos en donde la visión es controlada, como los juegos de estrategia en tiempo real, en comparación con los juegos de disparo en primera persona.

Existen técnicas para remover objetos de una escena entre las que se pueden nombrar a las siguientes:

- Culling de Angulo de Vista.
- Culling de Cara Trasera.
- Culling de Portal.
- Culling de Detalle.
- Culling de Oclusión.

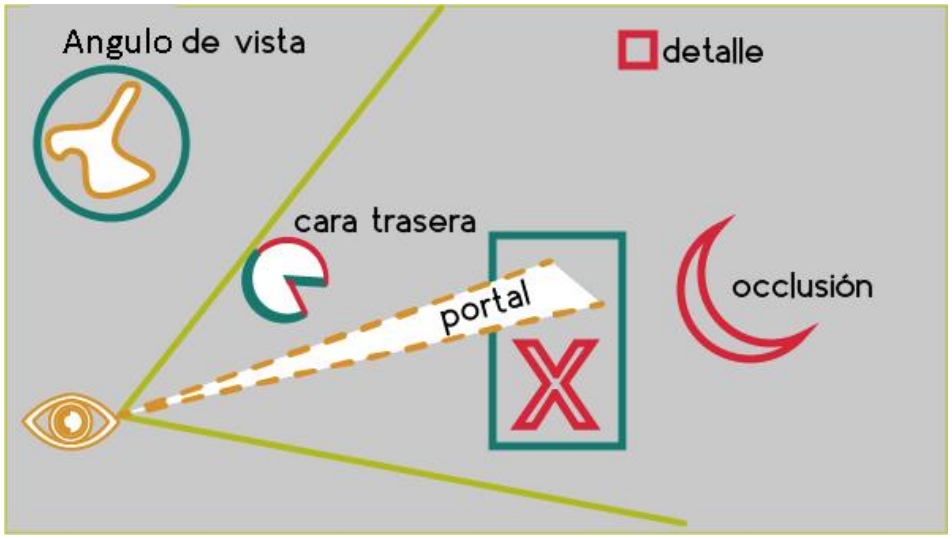


Figura 4. Culling

(Fuente: Game Rendering [Online] http://www.gamerendering.com/2008/11/02/basic-culling-techniques/)

3.7.1. Culling de Ángulo de vista

Todo objeto que se encuentre fuera del ángulo de visión de la cámara no será visible en la pantalla, por lo tanto puede ser no renderizado. Este método se realiza solo si la caja de colisión completa del objeto se encuentra fuera del campo de visión.

3.7.2. Culling de Cara trasera

Toda cara de un objeto que no se encuentre frente a la cámara no va a ser visible en la imagen final, por lo tanto no será necesario que se la renderice.
Este método en la actualidad viene implementado en las tarjetas de video debido a que es usado muy frecuentemente.

3.7.3. Culling de Portal

Esta técnica es muy usada para escenas en interiores, permite dividir una escena por medio de portales (puertas, gradas, etc). Al momento de la renderización la cámara se encontrará en uno de los cuartos y renderizará

normalmente lo que se encuentre dentro del foco de vista, y al momento de pasar por un portal recalculará y renderizará todo lo que entre en el nuevo foco de la escena dejado atrás la escena anterior.

3.7.4. Culling de Detalle

Cuando un objeto se encuentre muy lejos del punto de vista y no sea completamente visible, no es necesario dibujar todos los detalles del mismo o para economizar espacio en memoria puede ser no renderizado por completo.

3.7.5.Culling de Oclusión

Este método es el más complicado en su implementación. Permite que cuando se encuentren varios objetos uno atrás del otro, solo se renderice el que se encuentra más cerca al punto de vista, dejando a los objetos ocultos sin renderizar; esto se realiza mediante diferentes algoritmos, uno de los más utilizados es Z-buffer que organiza los objetos del más cercano al más lejano.

3.8. Retesselation

Técnica usada por la característica de TruForm de ATI que consiste en tomar un modelo basado en triángulos y transformarlo en uno de High-Order Surfaces para alisarlo y de nuevo pasarlo a un modelo de Triángulos [3].

3.9. Iluminación (lighting) Sombreado (Shading)

La luz interactúa con un objeto y el color que se ve es lo que queda del espectro después de haber sido absorbido por el objeto.

La iluminación de una escena se encarga de calcular la intensidad de luz en un punto en particular de una superficie mientras que el sombreado usa los cálculos de la iluminación para sombrear o dar matices a una superficie, objeto o escena.

Mediante todo esto se busca obtener una escena en tres dimensiones lo más real posible, implicando una mayor fidelidad en la representación de cada uno de los elementos que conformen la escena.

Este realismo dependerá de que como se maneje la simulación de iluminación y sombreado; los métodos de iluminación dan la intensidad y color de cada pixel del objeto mientras que los métodos de sombreado ayudan a reducir la cantidad de pixeles a los cuales se les aplicará un modelo de iluminación.

3.9.1. Características de la luz

Cuando las luces interactúan con un objeto se dan ciertos fenómenos, estos fenómenos ocasionan que la luz en el objeto pueda ser: (Ver Figura 5)

1. Total o parcialmente trasmitida
2. Total o parcialmente reflejada
3. Total o parcialmente absorbida

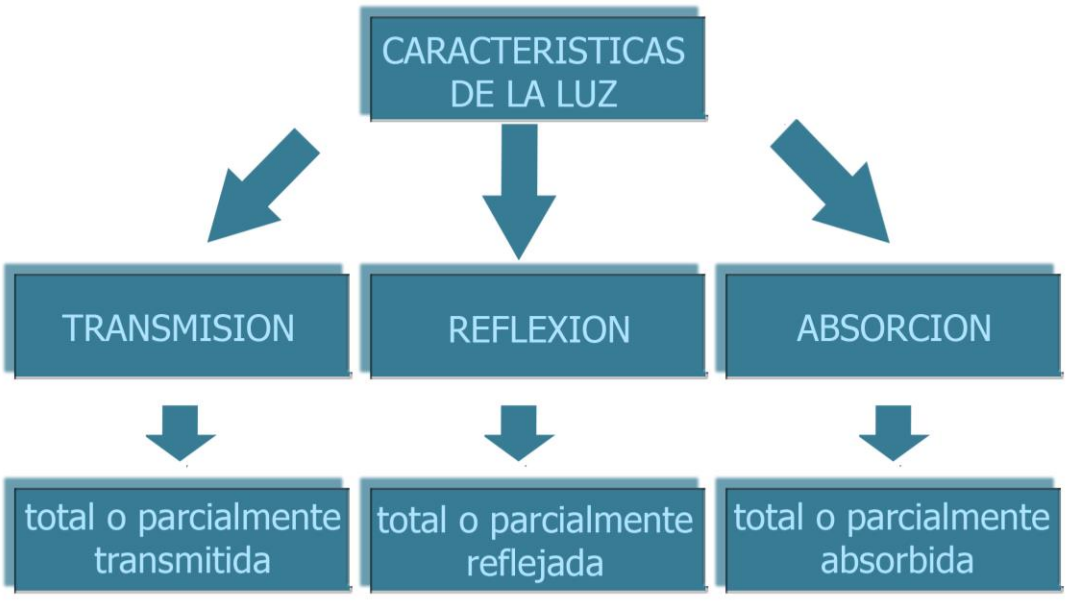


Figura 5. Características de la luz

(Fuente: Propia)

3.9.1.1. Transmisión

Un rayo de luz pasa por un objeto sin que esta sea esencialmente cambiada (se entiende así que el objeto es transparente), en el borde del objeto el rayo de luz transmitido (rayo incidente) cambia de dirección, a este efecto se lo conoce como refracción. De acuerdo al índice de refracción del material se produce alteraciones en la forma en la que se transmite la luz.

3.9.1.2. Reflexión

Cuando el rayo incide en un objeto que no transmite luz (objeto opaco) puede darse que la luz se difunda, se refleje o ambos fenómenos, esto depende en gran parte de la superficie del objeto.

Una superficie brillante o suave está hecha con partículas de índice de refracción igual, en estas superficies la luz se refleja a igual intensidad y ángulo que el rayo incidente.

La difusión o dispersión es un factor importante en la reflexión, cuando un objeto está compuesto por partículas de diferente índice refractivo un rayo de luz se dispersará dependiendo del tamaño de las partículas y los índices de refracción.

3.9.1.3. Absorción

La luz puede ser absorbida parcial o completamente por un objeto dependiendo de su pigmentación. El color final que se muestra es la longitud de onda del total de la luz que no ha sido absorbida.

3.9.2. Modelos de Iluminación

Los modelos de iluminación permiten simular el efecto que aplican las luces sobre las superficies. Estos modelos de iluminación pueden dividirse de la siguiente manera: (Ver Figura 6)

1. Modelos de Fuentes de Luz
2. Modelos de Reflexión
3. Modelo de Reflexión de Phong

Figura 6. Modelos de Iluminación

(Fuente: Propia)

3.9.2.1. Modelos de Fuentes de Luz

Definen la naturaleza que emana de una fuente de luz. En estos modelos solamente influyen ciertos elementos como: el material, las superficies en cuestión, el ángulo de incidencia de la luz y el ángulo con el eje visual (posición del observador). Se realizan por medio de cálculos simples.

Entre los modelos de fuente de luz contamos con los siguientes: (Ver Figura7)

1. Luz ambiente
2. Luz puntual o radial
3. Luz direccional (Spotligths)
4. Luz distante

3.9.2.1.1.Luz Ambiente

La luz no procede de un punto en concreto y es esparcida en todas las direcciones y produce un efecto de iluminación uniforme sobre cada punto de la escena. Su Luminancia se especifica mediante *Ia* que incluye 3 componentes que pueden definirse en un vector.

$$Ia = \begin{bmatrix} Iar \\ Iag \\ Iab \end{bmatrix} \quad [3.1]$$

La intensidad de iluminación *La* en un punto cualquiera en la superficie (p) es igual a la luminancia.

$$Ia = La \quad [3.2]$$

Aunque cada punto de la escena recibe la misma La cada superficie refleja la luz de manera diferente.

3.9.2.1.2. Luz Puntual o Radial

Este tipo de luz procede de un punto concreto (*po*) y emite rayos iguales en todas las direcciones. Por ejemplo la luz de un foco, una lámpara o una vela.
La Luminancia de una fuente de luz puntual se caracteriza por el siguiente vector.

$$I(po) = \begin{bmatrix} Ir(po) \\ Ig(po) \\ Ib(po) \end{bmatrix} \quad [3.3]$$

La intensidad de iluminación Lp en cualquier punto de la superficie (p) es:

$$Lp = fatt\, I(po, p) \quad [3.4]$$

Donde el factor de atenuación está dado por:

$$fatt = \frac{1}{a+bd+cd^2} \quad [3.5]$$

Siendo d la distancia entre el punto p y punto po.

a,b,c constantes definidas por el usuario para suavizar la iluminación puntual.

3.9.2.1.3. Luz Direccional (Spotlight)

Se construye a partir de una fuente de luz. Los rangos por los cuales se emite la luz son pequeños. Esta luz se dirige en una dirección concreta y se pueden controlar los valores del rango por el cual se emite, la intensidad de difusión, entre otros. Un ejemplo de luces direccionales son las luces de teatros, escenarios, etc. Se define con la siguiente fórmula:

$$Lp = fatt\ I(p, po)cos\theta \qquad [3.6]$$

Donde θ es el ángulo de abertura de la fuente de luz y

$fatt$ está dado en la fórmula [3.5]

3.9.2.1.4. Luz Distante

También conocida como fuente de luz paralela. Cuando la fuente de luz se encuentra muy distante al objeto, la luz llega al objeto sobre todos los puntos de una manera similar y no se necesita recalcular para cada punto del objeto, por lo tanto aunque la fuente de luz se encuentre cerca el objeto recibe los rayos de manera paralela simulando una fuente lejana. Un ejemplo de este tipo de luz es el Sol y como llegan los rayos a la Tierra. A pesar de que emite luz en todas las direcciones, al encontrarse tan distante la Tierra recibe los rayos como si fueran paralelos a ella. En coordenadas homogéneas una fuente de luz en el punto *p0* se representa de la siguiente manera:

$$po = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad [3.7]$$

Donde la fuente de luz distante se representa mediante el siguiente vector:

$$po = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} \qquad [3.8]$$

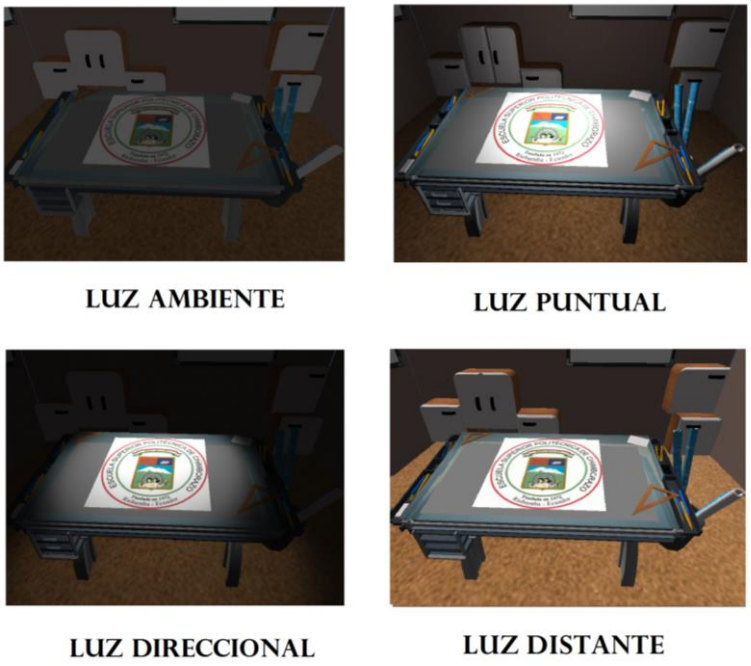


Figura 7. Modelos de fuentes de luz
(Fuente: Propia)

3.9.2.2. Modelos de Reflexión

Los modelos de reflexión se encargan de definir la intensidad de la luz reflejada desde una fuente de luz en la superficie de un objeto, teniendo en cuenta ciertas características esenciales y especiales de cada objeto. Estos modelos incluyen una serie de algoritmos y cálculos complejos que permiten la simulación de la luz reflejada.
Entre los modelos de reflexión principales contamos con los siguientes:

1. Reflexión de Ambiente
2. Reflexión Difusa o de Lambert
3. Reflexión Especular

3.9.2.2.1. Reflexión de Ambiente

La reflexión de ambiente indica que la intensidad de la luz de ambiente es la misma sobre cada punto de la superficie de todos los objetos de la escena.

Parte de la luz es absorbida y otra parte es reflejada. La cantidad que se refleja corresponde al coeficiente de reflexión de ambiente y la intensidad en el punto por lo tanto:

$$Ambiente = k_a L_a \quad [3.9]$$

Donde k_a es el coeficiente de reflexión de ambiente y

L_a es la intensidad de luz ambiente.

3.9.2.2.2. Reflexión Difusa o de Lambert

Para contar con una reflexión difusa o de Lambert los rayos de luz que se disparan hacia el objeto tienen ángulos levemente diferentes, debido a la rugosidad de las superficies lo que genera que no hay aun ángulo preferido de reflexión. Esta reflexión permite que la luz reflejada sea esparcida igual en todas las direcciones por lo que para todos los observadores será mostrada de la misma manera. La cantidad de luz reflejada depende del material del objeto debido a que se absorbe diferente cantidad de luz de acuerdo al material.

Esta reflexión es regida por la ley de Lambert que dice que:

$$R_d \propto \cos\theta \quad [3.10]$$

Donde θ es el ángulo de la normal n en un punto y la dirección de la luz es l. Por lo tanto el coseno del ángulo es igual al producto punto de n por l definiéndose de la siguiente manera:

$$\cos\theta = l.n \quad [3.11]$$

Resultando la fórmula de la reflexión difusa de la siguiente manera:

$$Difusa = L_d \; k_d \, max(l.n, 0) \quad [3.12]$$

Donde L_d es la intensidad del color de la luz difusa

k_d es el coeficiente de reflexión difusa y

$max(l.n, 0)$ da valores del $\cos\theta$ entre 0 y 1

3.9.2.2.3. Reflexión Especular

La reflexión especular le aporta “vida” a las imágenes ya que muestra un diferente color, reflejado de manera no uniforme, en cada punto de la superficie del objeto que a la vez varía de acuerdo al punto desde el cual se está observando.

Para calcular el color que se muestra en cada punto la fórmula dependerá de la superficie, el ángulo de reflexión y el ángulo concreto desde el cual se observa.

$$Especular = L_s\, k_s\, fmax(h.n, 0)s \qquad [3.13]$$

Donde L_s es la intensidad del color de la luz especular.

k_s es el coeficiente de reflexión especular.

f es una constante que toma el valor de 1 si el $\cos\theta$ es mayor que 0, caso contrario toma el valor de 0.

h es vector normalizado que se encuentra entre el punto de vista y la fuente de luz.

$max(h.n, 0)$ da valores del $\cos\theta$ entre 0 y 1.

s es una constante que muestra que tan brillante será la luz reflejada.

Mientras mayor sea este coeficiente menor será el brillo que se refleje.

3.9.2.3. Modelo de Reflexión de Phong

El modelo de reflexión de Phong es un modelo empírico de iluminación local que asigna brillo a los puntos de la superficie modelada sin tener en cuenta las interrelaciones entre objetos de la escena. El método se basa en el cálculo del brillo asignado a cada pixel del modelo de la superficie.

Describe como la superficie refleja la luz en una combinación de reflexión de ambiente de toda la escena, reflexión difusa para superficies rugosas y reflexión especular de superficies brillantes.

$$Phong = I_a + I_d + I_e \qquad [3.14]$$

Donde I_a es la reflexión de ambiente

I_d es la reflexión difusa y

I_e es la reflexión especular

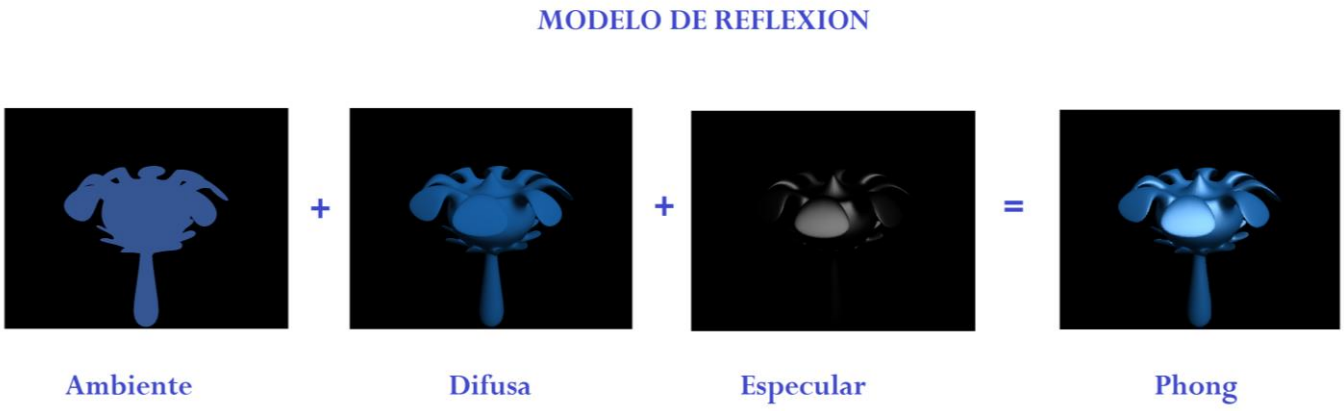


Figura 8. Modelo de Reflexión de Phong
(Fuente: Propia)

3.9.3. Modelos de Sombreado

Normalmente el cálculo de iluminación por cada pixel es redundante e innecesario por lo que se necesitan técnicas de sombreado que incluyan realismo y permitan acelerar el proceso de dibujado de escenas.

Existen varios modelos de sombreado entre los que por su complejidad o velocidad de procesamiento se destacan (Figura 9):

1. Modelo de Sombreado Plano o Constante
2. Modelo de Sombreado de Gouraud
3. Modelo de Sombreado de Phong

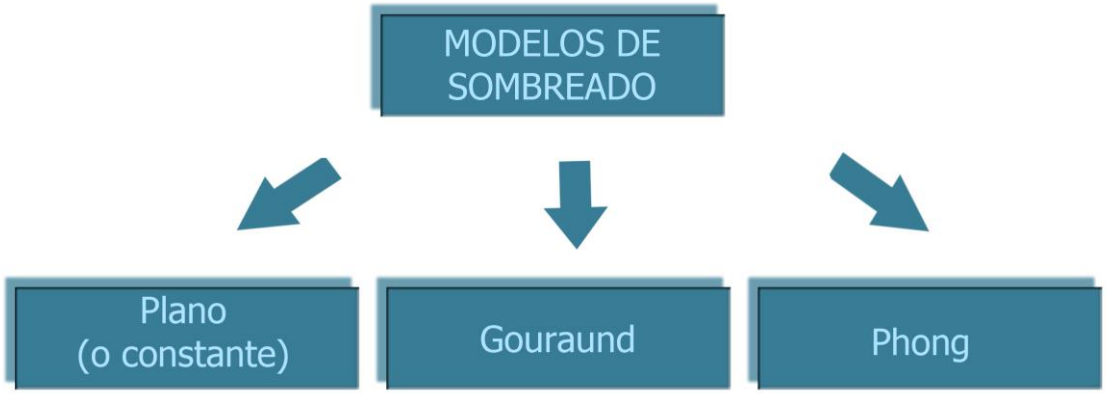


Figura 9.Modelos de Sombreado
(Fuente: Propia)

3.9.3.1. Modelo de sombreado constante o plano

Esta técnica funciona para objetos que tengan todas las caras completamente planas. Se encarga de calcular una única intensidad que se aplicará a todo el polígono.

El color de cada polígono dependerá de en qué parte de éste sea tomada la muestra, debido a que con una misma normal en distintos puntos el modelo de iluminación devolverá colores diferentes.

El sombreado constante o plano es válido si se tiene una fuente de luz lejana y constante para todo el polígono el observador esta infinitamente lejano y constante y se modela un polígono que no es una aproximación de curva.

3.9.3.2. Modelo de Sombreado de Gouraud

Es una técnica de sombreado suave que se encarga de simular efectos de luz y color sobre la superficie de los objetos.

Calcula los valores de luminosidad en cada vértice por medio del promedio de los valores de las superficies que convergen hacia dicho punto, este cálculo lo realiza por medio del modelo de reflexión de Phong. Se procede al cálculo de

las intensidades de cada uno de los pixeles de las superficies implicadas por medio de la interpolación bilineal de los valores estimados de los vértices.

3.9.3.3. Modelo de Sombreado de Phong

Esta técnica consiste en la interpolación en cada pixel en base al cálculo de las normales en cada vértice, para asignar un color a cada uno de estos pixeles calculándolo en base a la normal interpolada y al método de iluminación.

Mediante esta técnica que es más precisa que la de Gouraud, se reproduce mejor la reflexión especular y se restaura la curvatura de los objetos poligonizados. Esta técnica ofrece resultados mucho mejores en cuestión de suavizado de texturas, aunque computacionalmente es más costosa que los otros métodos dado que para cada pixel se calculará el método de iluminación.

3.9.4. Generación de Mapas de Luz (Light Map)

Es una estructura de datos que contiene el brillo de las superficies de gráficos 3D que se agregan a una escena para darle efecto de iluminación a todos los elementos.

Los Light Map son usados para objetos estáticos y los cálculos son realizados previamente y almacenados en bitmaps, lo que permite que a una escena creada se agreguen luces consiguiendo un efecto de iluminación a nivel de pixel en tiempo de ejecución.

3.9.5. Textura

Una textura es una imagen que se usa para cubrir una superficie de un objeto 3D para cambiar su apariencia y adicionar detalle sin aumentar su complejidad geométrica (Ver Figura 10).

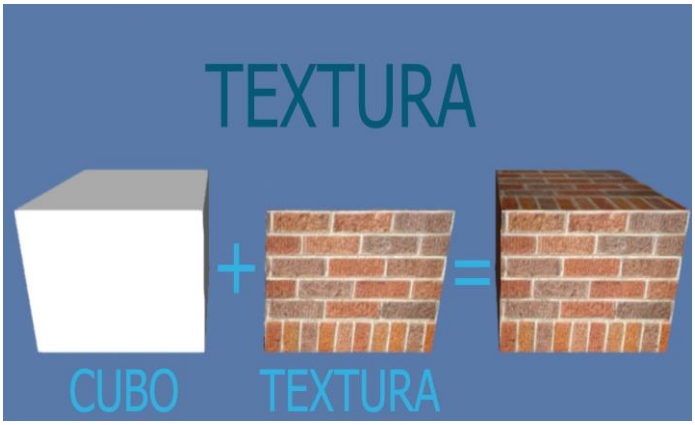


Figura 10. Textura
(Fuente: Propia)

La textura personaliza los planos y los polígonos teniendo en cuenta la forma en que se aplica el método de sombreado y de iluminación, dando una sensación tridimensional que por medio de los sentidos ayuda a tener una mejor experiencia de una escena.

3.9.6. Texturas Múltiples

Una textura múltiple (Ver Figura 11) Implica el uso de más de una textura a la vez en un polígono. Incluyendo el uso de una imagen sobre otra con distintas transparencias. Estas múltiples texturas implican múltiples renderizados para poder lograr un resultado óptimo. Por lo tanto se necesita el uso de un acelerador grafico para que el mapeo sea de mejor calidad.

Figura 11. Texturas Múltiples
(Fuente: Propia)

3.9.7. Mapeado topológico

Mediante el Mapeado topológico o también conocido como bump mapping, se añade color a un objeto y se busca dar un aspecto rugoso a las superficies del mismo que le den la apariencia de relieve logrando grandes parecidos con efectos naturales.

El bump mapping altera la orientación de las normales de una superficie durante el proceso de sombreado pero no cambia la topología ni la geometría de las mismas. (Ver Figura 12)

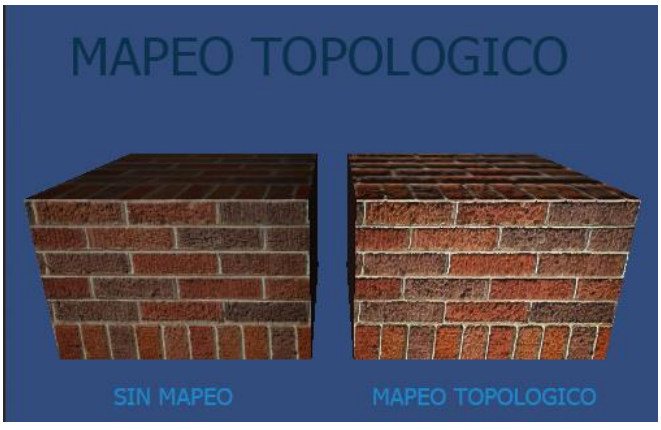


Figura 12. Mapeo Topológico
(Fuente: Propia)

3.9.8. Mapas MIP

Los mapas MIP, también conocidos como mipmaps, son bitmaps que contienen información de múltiples copias de la misma textura pre procesadas con la característica de que cada una de las copias es de menor tamaño y resolución.

Se utiliza la textura principal cuando se necesita un renderizado a todo detalle y dependiendo de la distancia y ubicación del objeto se va utilizando cada uno de los otros elementos del mipmap y se procede a interpolar los rangos más cercanos en caso de que la escena se renderice a un tamaño diferente a los almacenados.

3.9.9. Antialiasing

Permite minimizar el efecto causado por señales contiguas distintas (aliasing) por medio de la eliminación de frecuencias muy elevadas que no pueden ser representadas en la resolución del monitor. Mediante esta técnica se busca que los bordes no se vean como pixelados o dentados; para esto pueden utilizarse dos técnicas, la primera difumina los vértices y bordes de los polígonos, mezclándolos y sobreponiéndolos uno delante de otros, o la segunda se consigue quitando los bordes de todo el marco lo que implicaría el uso de más memoria. (Ver Figura 13)

Figura 13. Antialiasing
(Fuente; Propia)

3.10. Sistemas de Scripts

Los sistemas de Scripts son lenguajes diseñados para ejecutarse por un intérprete en tiempo real; dentro de él son leídos línea a línea en lugar de ser compilados todo en conjunto, lo cual vuelve más eficiente a los programas.
Este tipo de lenguajes permiten una flexibilidad adicional a los programas, facilidad de depuración y mayor rendimiento.

Estos lenguajes también conocidos como lenguajes interpretados son independientes de la máquina y del sistema operativo, debido que contienen llamadas a funciones conocidas por el intérprete en lugar de instrucciones para un procesador específico. Además permiten la modificación en tiempo de ejecución lo que ayuda a no tener que compilar una aplicación cada vez que se realice un cambio.

La desventaja de este tipo de lenguajes en función a los lenguajes compilados es el tiempo que les toma para ser interpretados, para esto algunos de estos lenguajes utilizan una máquina virtual que se encarga de traducir a un lenguaje intermedio para que sea más rápida la conversión a lenguaje de bajo nivel.

3.11. Sonido

La música está presente en los video juegos desde que ellos existen. En sus principios contaban con pocos bits que permitían reproducir sonidos como un beep con cada movimiento del jugador, y con el paso de los años han ido evolucionando hasta llegar a los sonidos realistas con los que se cuenta en la actualidad.

Este trabajo existente por años ha creado resultados como mejor música o sonidos más realistas y de mejor calidad que se acercan más a las situaciones reales, logrando con esto generar emociones en el usuario, para que así la experiencia que tenga sea más realista.

Según Karen Collins en su libro Game Sound: An Introduccion to the History, Theory, and Practice of Video Game Music and Sound Design, [4] el audio para un video puede dividirse en 4 grupos como se muestra en la figura 14 y se describe posteriormente.

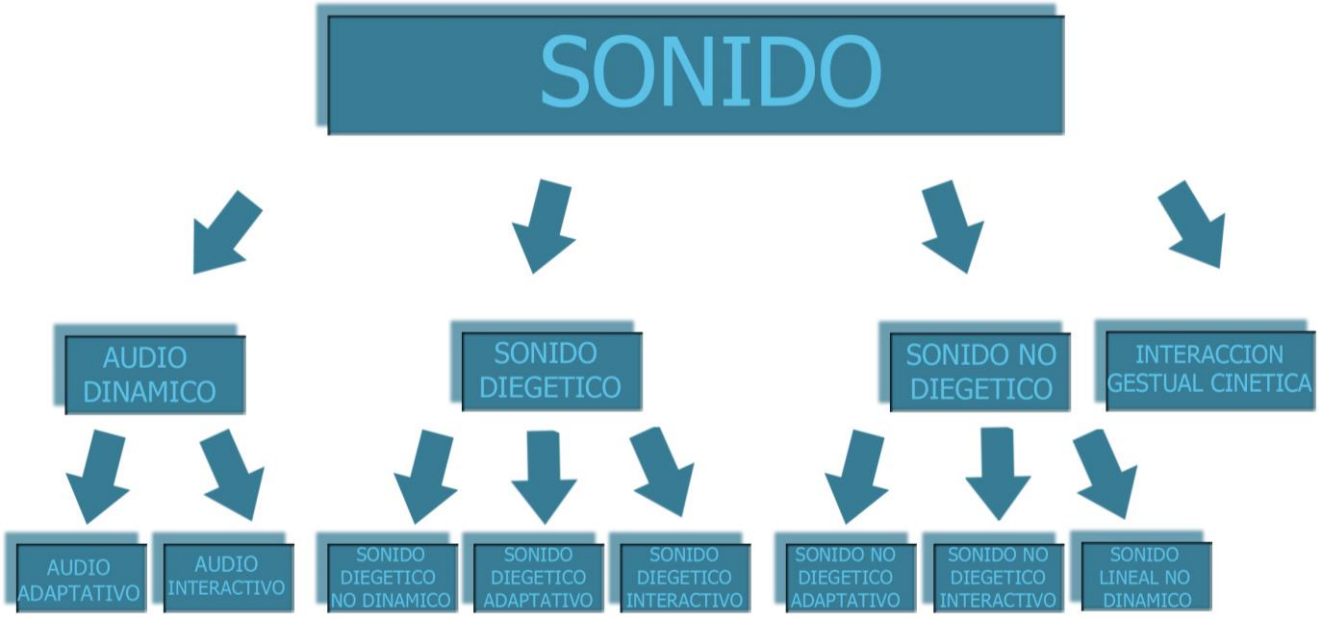


Figura 14. Sonido
(Fuente: Propia)

3.11.1. Audio Dinámico

Por audio dinámico se entienden sonidos que reaccionan a los cambios que se den en el juego por interacción del usuario o por necesidad del mismo juego. Dentro del audio dinámico existe una sub clasificación que incluye al Audio Adaptativo y al Audio Interactivo.

Audio Adaptativo: Es el sonido que reacciona al ambiente del juego sin importar los comandos que el usuario ejecute.

Audio Interactivo: El audio Interactivo responde a las entradas del usuario y repercute en el sonido final del juego.

3.11.2. Sonido Diegético

Son sonidos reales que se producen al alcance del oído del usuario, como los efectos de sonido o el mismo dialogo.

Sonido Diegético no dinámico: Son sonidos que se producen al alcance del oído del usuario sin que este se involucre directamente en ellos.

Sonido Diegético Adaptativo: Este sonido reacciona debido cambios en el entorno del juego pero no a acciones que el jugador haya realizado directamente.

Sonido Diegético Interactivo: Estos sonidos ocurren en el área en la que el jugador puede interactuar, por ejemplo los pasos o los impactos por disparo de arma.

3.11.3. Sonido No-Diegético

Este tipo de sonido hace referencia a los efectos de sonido y a la música de fondo en los juegos.

Sonido no Diegético Adaptativo: Son los sonidos que se generan como reacción a lo que ocurre en el juego, pero no son afectados por lo que haga el usuario, no son parte del entorno ni son escuchados por otros personajes del juego; Por ejemplo la explosión de un avión por un choque.

Sonido no Diegético Interactivo: Estos sonidos pueden ser afectados por el juego o el jugador pero no son escuchados por los otros personajes; Por ejemplo la música interactiva o los sonidos de efecto que reaccionen a como se está desarrollando el juego.

Sonido Lineal no Dinámico: Son sonidos que no son afectados por ninguna entrada de los jugadores o desarrollo del juego. Este tipo de sonidos son usados frecuentemente en escenas ininterrumpidas; por ejemplo el sonido de fondo de las escenas de un juego.

3.11.4. Interacción Gestual Cinética

Por interacción gestual cinética se entienden los sonidos diegéticos o no diegéticos en los que el jugador interactúa físicamente con el sonido en pantalla. Por ejemplo el uso de un controlador que permita tocar un instrumento en pantalla o controlar la velocidad de las armas u otra acciones que generen sonidos.

Procesamiento del Sonido

El sonido se procesa de manera similar a los modelos, en muchas ocasiones un software es el que lo procesa antes de que llegue al procesamiento hardware. A este proceso se lo conoce como premezcla en software.

Para el manejo del sonido dentro de un videojuego existen dos formas:

1. El manejo de los archivos en formato .wav (o formatos similares) que requiere de un amplio espacio en memoria para emitir un sonido de buena calidad.

2. El uso de archivos midi, con estos archivos la cantidad de memoria necesaria se reduce, pero la calidad de sonido obtenida es inferior.

3.12. Inteligencia Artificial (IA)

Se conoce como Inteligencia Artificial a las técnicas que en conjunto dan al videojuego la ilusión de comportamiento inteligente en todos los personajes del juego que no sean controlados por un jugador, en contraste a los personajes que son manejados por él y se controla su comportamiento.

La inteligencia de NPC (siglas en ingles de Personaje No Jugador) no implica que juegue mejor que los humanos, sino que aprenda sus debilidades y

fortalezas para que la experiencia del juego sea más realista y atractiva para los usuarios.

La Inteligencia Artificial es muy usada en juegos de estrategia, juegos en tiempo real, etc ya que permite que los NPC “vayan aprendiendo” del entorno y los otros jugadores puedan navegar fácilmente, realicen búsqueda de rutas, adaptar su comportamiento, etc.

Otra de las aplicaciones que en la actualidad el campo de la IA ha desarrollado es el conocido como “instinto de supervivencia”, que permite que el NPC “pueda aprender” del ambiente y reconocer si cada elemento es perjudicial o beneficioso para él de tal manera que puede tomar decisiones y realizar acciones que le permitan defenderse o atacar. Por ejemplo un Avatar en un ambiente de guerra.

3.12.1 Búsqueda de caminos

La búsqueda de caminos obedece a exigencias surgidas con el avance tecnológico. Estas búsquedas pretenden simular la inteligencia humana para llegar de un lugar a otro, tomando en cuenta aspectos importantes como el tiempo que esto implique, la distancia a recorrer, la ruta que se designará entre otros.

Para la búsqueda de caminos existen varios algoritmos que de acuerdo a las necesidades devuelven respuestas diferentes; por ejemplo ciertos algoritmos se encargarán de encontrar el camino más corto, el camino más rápido, o de asignar valores a los caminos en función de variables establecidas (costo) para así llegar a seleccionar el mejor, etc.

Para realizar la búsqueda de caminos se parte de la asignación de valores a los nodos, es decir que cada lugar en el mapa debe ubicarse.

Los algoritmos de búsqueda más sencillos parten de un mapa en el cual se encuentran todos los nodos (conocido como árbol) que será donde se realizará la búsqueda; para esto se partirá siempre desde el primer nodo del árbol y se buscará secuencialmente nodo por nodo en amplitud o en profundidad de

acuerdo al algoritmo aplicado. Para evitar que cada vez que se realice una búsqueda se regrese al nodo inicial y se recorran uno por uno, ciertos algoritmos asignan conocimiento heurístico a cada uno de los nodos; de esta manera la búsqueda será más sencilla y dependerá de factores como el costo, el destino, entre otros.

Para la programación de video juegos y mundos virtuales, y más si éstos van a encontrase en la Web, se necesitan respuestas rápidas; es por esto que el algoritmo más utilizado es A*, ya que éste es flexible y ha sido optimizado para utilizar información a priori para hacer la búsqueda, de esta forma trata de encontrar el camino directamente entre dos puntos del mapa.

El valor del camino encontrado se da aplicando la siguiente fórmula:

$$f = g + h \qquad [3.15]$$

Donde g representa el costo exacto que implicará llegar del nodo inicial hasta el nodo al que deseamos llegar.

h representa el costo estimado para llegar del nodo actual al nodo buscado, se usa la letra h por ser un valor heurístico ya que se debe tomar en cuenta que el valor exacto no se conoce aún y el valor que se asigne es una aproximación. Mientras más exacta sea la aproximación mejor será el resultado.

f representa el mejor camino encontrado en función a g y a h. Mientras más bajo sea el valor de f se deduce que el recorrido será más corto y más rápido.

A* trabaja con dos listas adicionales, la una conocida como Abiertos y la otra llamada Cerrados. La lista Abiertos contiene los nodos que aún no han sido explorados, mientras que la lista Cerrados contiene los nodos que han sido explorados. Se considera como explorado a un nodo si el algoritmo ha calculado el valor de f para cada uno de los nodos que se conectan a este. Esto permite que un nodo pueda ser visitado nuevamente después de encontrase en un nodo que se ubique delante de él.

Pseudocódigo de A*

Se detallará el Pseudocódigo del algoritmo A* tomado del Capítulo “Búsqueda de Caminos con A*” del libro AI Game Programming Wisdom [5]

1. P = punto de inicio.
2. Se asignan los valores de f, g y h a P.
3. Agregar P a la lista Abiertos. En este momento P es el único nodo en la lista Abiertos.
4. B = el mejor nodo (el que tiene el menor valor de f) de la lista de Abiertos.
    a. Si B es el nodo de llegada, entonces fin y el camino se ha encontrado.
    b. Si la lista Abiertos está vacía, entonces fin y no se puede encontrar un camino.
5. C = un nodo válido que se conecte con B.
    a. Se asignan los valores de f, g y h a B.
    b. Verificar si C se encuentra en la lista Abiertos o Cerrados.
        i. Si está en la lista Cerrada, verificar si el nuevo camino es más eficiente (es decir tiene menor valor de f).
            i. Si se cumple la condición, actualizar el camino.
        ii. Caso contrario, añadir C a la lista Abiertos.
    c. Repetir el paso 5 para todos los hijos válidos de B.
6. Repetir el paso 4.

CAPITULO IV

DESCRIPCIÓN DE HERRAMIENTAS Y MOTORES DE JUEGOS EXISTENTES EN EL MERCADO

Con un sólido marco teórico del campo de estudio se procede a dar una descripción de las principales herramientas de animación y modelado así como de los motores de juegos existentes en el mercado; para la posterior selección de los más adecuados que se utilizarán para demostrar la metodología.

4.1. Descripción de herramientas de modelado y animación

4.1.1. Blender

Blender es una de las aplicaciones de Software Libre para gráficos 3D más populares en el mundo, por ser un programa muy completo a pesar del pequeño tamaño que ocupa y la capacidad de funcionamiento en equipos que cuentan con pocos recursos en hardware. Desde sus inicios, fue concebida como una aplicación multiplataforma integrada para la creación, modelado y animación de un diverso rango de contenido 2D y 3D orientado a tareas.

Blender cuenta con una interfaz robusta, altamente optimizada y con una gran cantidad de botones y menús para la producción de gráficos 3D. Pese que para algunos técnicos su interfaz es criticada por ser de primera poco intuitiva por no usar un sistema de ventanas como la mayoría de software existente, pero con el empleo de su interfaz, este programa se vuelve muy amigable. Sus principales ventajas son la distribución de menús que cuenta con una configuración personalizada y la vista de cámaras.

Esta herramienta cuenta con un motor interno de juegos 3D en tiempo real, lo que permite la creación de contenido interactivo que puede ser reproducido de manera independiente. Entre las funcionalidades incluidas en el paquete se encuentran: modelado, texturizado, iluminación, animación, post-procesamiento de video además de escultura y pintura digital.

Blender fue desarrollado por la empresa Not a Number, como software libre con su código disponible bajo la licencia GNU GPL, cuenta con una comunidad de soporte en la Web y en coordinación con The Blender Foundation en Holanda que se encarga de un constante desarrollo.

Actualmente es compatible con todas las versiones de Windows, Mac OS X, Linux, Solaris, FreeBSD e IRIX.

Blender en la computación gráfica:

Blender es una herramienta que cuenta con una gran aceptación en la industria de la animación especialmente por animadores independientes, a pesar de ser una herramienta considerablemente nueva.

Varios proyectos han usado esta herramienta profesionalmente como:

- Plumíferos: Un largometraje animado del 2010 realizado totalmente con software libre y principalmente Blender.
- Spider-Man 2: es usado para la pre visualización de escenas.
- Friday or another day: Se producen e integran gráficos mediante Motion Track, entre otros.

Características:

- Software libre.
- Disponible para Windows, Linux y Mac OS.
- Suite de creación completamente integrada, incluye una gran cantidad de las herramientas necesarias para la creación de contenido 3D como herramientas de modelado, mapeo uv, texturización, rigging, skinning, animación, simulaciones, integración de partículas, renderizado, post-producción, creación de juegos entre otros.
- Arquitectura 3D de alta calidad que permite una creación rápida y eficiente de trabajo.
- Comunidad de usuarios brindando soporte por medio de foros y conversaciones.
- Facilidad de distribución.
- Edición de audio y sincronización de video.
- Capacidad para una gran variedad de primitivas geométricas, incluyendo curvas, mallas poligonales, vacíos, NURBS.
- Acepta formatos gráficos como TGA, JPG, Iris, SGI, o TIFF.
- Características interactivas para juegos como detección de colisiones, recreaciones dinámicas y de lógica.

4.1.2. Cinema 4D Studio

Cinema 4D Studio es una herramienta de software propietario que en la actualidad cuenta con una versión gratis para estudiantes, utilizada para la creación de animaciones y gráficos avanzados en 3D, considerada como una herramienta de apoyo que permite la fácil y rápida creación de gráficos para los diseñadores. Esta herramienta permite la creación de modelado de procedimientos y de polígonos, renderizado de muy alta velocidad, animación, iluminación, texturizado.

Cinema 4D es muy fácil de manejar y perfeccionar su uso lleva muy poco tiempo. Posee una interfaz y herramientas intuitivas, personalizables y fáciles de usar, lo que la ha convertido en una de las herramientas de animación y modelado favoritas para la creación de gráficos en movimiento. La aplicación incluye herramientas avanzadas usadas para render, character, creación de pelo con movimiento, creación de rigs, animación avanzada de personajes y cuenta con un motor de físicas que permite representar la interacción y colisión de objetos rápidamente solamente configurando las opciones.

Cinema es una herramienta modular, lo que permite que partiendo de una versión básica se puedan agregar componentes independientes que sean especializados en funciona a las necesidades con las que se cuente. Entre los módulos principales utilizados se cuenta con: Modulo de Renderizado, Modulo de Dinámicas, Huesos, Pelo, Partículas, Animación Compleja.

Originalmente fue creada por la empresa alemana Maxon para ser utilizado por Commodore Amiga, en un futuro se exporto para ser usada en sistemas operativos Windows y Mac OS.

Características:

- Software propietario.
- Versión libre con muchas limitaciones para estudiantes.
- Disponible para Windows y Mac OS.
- Facilidad de creación de rigeado.
- Herramientas avanzadas para animación de personaje.
- Herramientas para render de alta calidad con mínimo uso de memoria.
- Simulación de interacción para cuerpos rígidos y blandos.
- Creación de movimiento realista al establecer parámetros básicos.
- Sistema para efectos de partículas inteligentes.
- Herramientas para clonación en tiempo real.
- Características de iluminación avanzadas.

- Soporta una gran cantidad de formatos de archivo.
- Permite crear funciones propias por medio de lenguajes de scripteo.
- Entre otras.

4.1.3. Autodesk 3DS Max

Autodesk 3DS Max es una herramienta propietaria muy utilizada en juegos, películas y por artistas especializados por considerarse completa debido a la facilidad que presenta para el modelado, renderizado, composición de escena, creación de gráficos y animación 3D, además de nuevos componentes para el manejo de partículas, multitudes y perspectiva.

3DS Max, o como era conocido en sus inicios 3D Studio Max, fue creado por el Grupo Yost para luego pasar a manos de Autodesk quien continuó con su desarrollo. La primera versión que salió a la venta fue para DOS con el nombre 3D Studio, contaba únicamente con 5 módulos y tenía una comunidad de usuarios para soporte y ayuda por atrás impulsada por sus creadores. Cuando se reescribió para Windows se le agregó la palabra MAX al nombre para no perder la marca utilizada en DOS y al cambiar de nombre la división multimedia de Autodesk a Kinetix el nombre completo del programa fue Kinetix 3D Studio MAX. En el año 2000 la división Kinetix cambio de nombre por lo que la herramienta adoptó el nombre que hasta la actualidad es utilizado: Autodesk 3DS Max.

Características:

- Software propietario.
- Disponible para Windows.
- Funciones integradas para animación 3D de alta calidad.
- Compatibilidad con mapas vectoriales.
- Creación eficaz de objetos paramétricos.

- Creación de personajes verosímiles.
- Simulación de fluidos.
- Compatible con sombreados.
- Interfaz de usuario configurable.

4.1.4. Autodesk Maya

Autodesk Maya es una herramienta de software propietario muy utilizado para la animación 3D que incluye tecnología de última generación y nuevas herramientas para la realización de renderización, modelado, simulación de fluidos, simulación de otros elementos, composición, animaciones 3D por computador, rastreo de movimiento, etc.

Esta herramienta es robusta y presenta posibilidades de personalización de herramientas e interfaz; trabaja en cualquier tipo de superficie como paramétricas, polígonos y subdivisión de superficies.

El lenguaje que utiliza es Maya Embedded Language conocido como MEL, que permite la creación de scripts y personalizar el paquete.

Maya surge de la evolución y fusión del código de las herramientas Power Animator y The Advanced Visualizer tras la fusión de las empresas encargadas del desarrollo de cada uno de ellos. Después de varios años la nueva empresa creada es absorbida por Autodesk, es por eso que se complementa su nombre al actual Autodesk Maya.

En sus inicios se comercializaba en dos versiones, Maya Complete una versión básica con los módulos más importantes como modelado, animación, renderizado; y Maya Unlimited que era una versión avanzada que incluía los módulos anteriores más módulos de manejo de fluidos, esta versión tenía un precio más elevado que Maya Complete pero resultaba similar al de otras

herramientas de la época. A partir del 2011 las dos versiones se fusionaron creando una sola versión de Maya.

A pesar de ser una herramienta propietaria existe una versión gratuita para uso no comercial llamada Maya Personal Learning, todos los trabajos que se realizan en esta versión tienen un logo en marca de agua de la compañía lo que ayuda para que no sean distribuidas.

Este software ha tenido un amplio impacto en la industria del cine debido a que es una herramienta de efectos visuales ampliamente usada por su facilidad de personalización y ampliación, lo que le ha hecho formar parte de películas ganadoras del reconocido premio Oscar como Harry Potter, Hugo, Kung Fu Panda, entre otros.

Características:

- Software propietario.
- Disponible para Windows, Linux y Mac OS.
- Abierto a software de otros desarrolladores.
- Altamente personalizable.
- Herramientas de animación por cuadros, procedimientos y secuencias.
- Modelado de mallas de polígonos y superficies.
- Gran cantidad de renderizadores integrados.
- Editor de nodos.
- Simulación de efectos de fluidos.
- Integración 2D Y 3D.

4.1.5. Comparación de herramientas de modelado

Para el presente estudio es necesaria la comparación de las herramientas de modelado más significativas que existen en el mercado para seleccionar la herramienta que mejor se adapte a lo que necesita el proyecto (Ver Tabla VI.I).

TABLA IV. III. COMPARACION DE HERRAMIENTAS DE ANIMACION Y MODELADO

	 **CINEMA 4D**	 **AUTODESK MAYA**	 **AUTODESK 3DS MAX**	 **BLENDER**
PRECIO	$3,495	Versión pagada $3,675. Maya cuenta con una versión gratuita estudiantil	$3,675	Gratis
SISTEMA OPERATIVO	Windows, Linux	Windows , Mac OS, Linux	Windows	Windows,Mac OS,Linux
USO PRINCIPAL	• Animación • Iluminación • Modelando • Efectos visuales 3D • Renderizado • Simulación	• Modelado • Animación (Vídeo) • Iluminación • Renderizado • Efectos visuales 3D	• Modelado • Animación • Video Juegos • Iluminación • Renderizado	• Animación • Iluminación • Modelado • Renderizado • Efectos visuales 3D • Sculpting
CARACTERÍSTICAS DE MODELADO	• Primitives • Polygons • Subdivision Surfaces • Soft Selection • NURBS • Sculpting Brush • Bezier Curves	• Primitives • Polygons • Subdivision Surfaces • Soft Selection • NURBS • Sculpting Brush • Bezier Curves	• Primitives • Polygons • Subdivision Surfaces • Soft Selection • NURBS • Sculpting Brush • Bezier Curves	• Primitives • Polygons • Subdivision Surfaces • Soft Selection • NURBS • Sculpting Brush • Bezier Curves
FACILIDAD DE USO	• 93.8% • Interfaz de usuario personalizable	• 93.8% • Interfaz de usuario personalizable • Menus extraibles	• 90% • Interfaz de usuario personalizable	• 80% • Interfaz de usuario personalizable
LÍMITE DE POLÍGONOS TRABAJABLES	• 1,000,000	• 1,000,000	• 1,000,000	• 1,000,000
FACILIDAD DE MODELADO	• 87%	• 100%	• 100%	• 92.5%
MANEJO DE TEXTURAS Y MATERIAL	• 87.5%	• 100%	• 93.8%	• 90%
ANIMACIÓN	• 81.3%	• 100%	• 100%	• 87.5%

Fuente: Propia.

4.2. Descripción de motores de videojuegos

4.2.1. Unity 3D

Unity 3D, o también conocido simplemente como Unity, es un motor gráfico 3D multiplataforma, muy versátil, utilizado mayormente para la creación de videojuegos. Es considerada como una de las herramientas más poderosas existentes en la actualidad ya que su motor de renderizado esta cien por ciento integrado a un conjunto de herramientas intuitivas y permite la rápida creación de contenido interactivo tanto 2D como 3D.

Esta herramienta diseñada por Unity Technologies cuenta con dos tipos de licencias, la primera es una gratuita que cuenta con ciertas limitaciones y la otra conocida como Unity-Pro que es pagada. En la actualidad se encuentra disponible para el desarrollo en los sistemas operativos Windows y Mac OS, y permite publicar sus productos en múltiples plataformas. Se puede publicar para PC en los sistemas operativos: Windows, Linux y Mac OS; para dispositivos móviles: Android, Blackberry, Iphone, Windows Mobile; y para consolas de juegos: Nintendo, Wii, Xbox 360, Play station 3, PS Vita. Para publicar en la Web se utiliza un complemento llamado Unity Web player que es gratuito.

Unity utiliza un editor que es considerado el eje central en la construcción de toda aplicación, cuenta con un editor visual lo que permite que el programador vaya construyendo el juego y por otro lado se programa el comportamiento de los assets por medio de un lenguaje de scripteo que usa una versión de Java Script y C #, o una variante de Python. Varios de los miembros de la comunidad de soporte han desarrollado editores que se encuentran disponibles gratuitamente. Unity permite crear los ejecutables para cada una de las distribuciones solamente con elegir el sistema operativo y la arquitectura.

Unity 3D cuenta con 5 elementos o vistas como puede ver en la Figura 15 y su interfaz puede ser modificada de acuerdo a las necesidades del usuario.

Permite construir la parte visual de la aplicación en la Vista de Escena que ofrece un entorno 3D donde se pueden posicionar objetos importados, rotarlos, escalarlos, moverlos todo esto sin salir de esta vista y automáticamente previsualizarlos en la Vista de Juego que permite reproducir la aplicación que está creándose para probar su funcionamiento.

En la vista de Proyecto se encuentran todos los assets del proyecto normalmente ordenados en carpetas, es acá a donde se importarán o crearán los elementos de la escena como objetos y texturas y los scripts para el funcionamiento; todo esto permite el manejo de los elementos en la escena solamente arrastrándolos.

La Vista de Jerarquía contiene los objetos que se encuentran en un momento dado en la escena. En esta vista se nos facilita encontrar todos los elementos de la escena sin la necesidad de moverse por toda ella. Por último el Inspector se encontrará la información detallada de los assets de la escena.

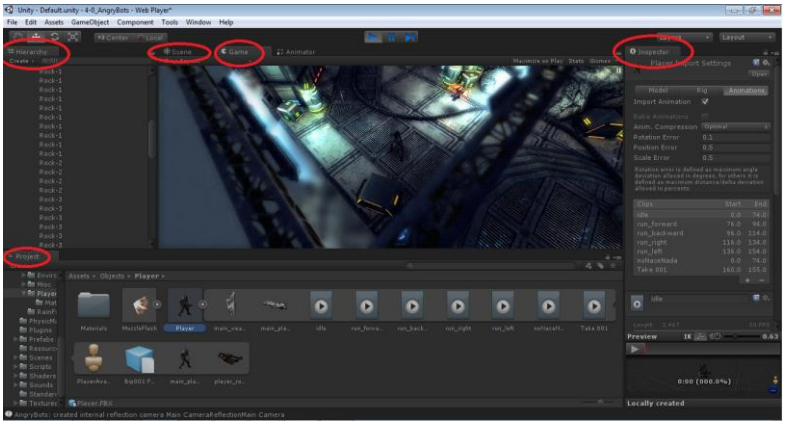


Figura 15. Interfaz de Unity
(Fuente: Propia)

Características:

- Software libre en su versión básica.
- Disponible para Windows y Mac OS.
- Espacio de trabajo intuitivo y de fácil manejo con opciones para Ejecutar, Probar y Editar de una manera más rápida.
- Alta calidad visual y de sonido, mezclando en tiempo real gráficos 3D y audio.

- Poderoso sistema de animación que hace que las escenas se vean fluidas.
- Multiplataforma, permite publicar en varias plataformas y para varios tipos de dispositivos fácilmente.
- Soporta assets que pueden exportarse desde múltiples herramientas de modelado y se importan automáticamente.
- Existe una gran cantidad de documentación.
- Cuenta con una comunidad de miembros que brindan soporte.

4.2.2. Unreal Engine

Unreal Engine es un motor de juegos 3D conocido por algunos como el más avanzado del mundo por la gran cantidad de premios que ha ganado. Presenta una gran cantidad de nuevas herramientas profesionales, intuitivas y de fácil manejo que permiten renderizado, digitalización, visualización, entre otros con el fin de crear y desarrollar video juegos de escritorio y móviles.

Esta herramienta creada por la empresa Epic Games que permite desarrollar solamente en Windows es multiplataforma al estar diseñado para DirectX (plataforma Windows) y OpenGL (plataformas Linux y MacOS) además es compatible con consolas como Wii, Xbox 360, Gamecube, PlayStation 4.

Es una de las herramientas más utilizadas en el mundo por grandes empresas como Microsoft Stuidos, Gearbox, Ninja Theory formando parte del desarrollo de varios juegos muy importantes como Star Wars, Batman o Gear Wars.

Una de las características principales de Unreal es el gestor de contenido que permite a los usuarios ubicar assets fácilmente aunque no estén cargados en la escena. Unreal cuenta además con una herramienta que ofrece soluciones de minería de datos, para tomar decisiones en base a estadísticas, lo que presenta facilidad para el desarrollo de juegos de multijugador en línea.

Otra de las características importantes es un complemento para el manejo de luces, principalmente la iluminación global, sin la necesidad de una herramienta externa.

Unreal permite la edición del código en C++ y el manejo del código a bajo nivel. En la actualidad la empresa creadora se encuentra trabajando en realidad virtual buscado la optimización del software en visores de realidad aumentada. Unreal Engine cuenta a partir del 2009 con una versión gratuita limitada y no comercial conocida como Unreal Development Kit o UDK, que permite la creación de juegos para sistemas operativos Windows, Mac OS y para Iphone y Ipad.

Características:

- Software propietario.
- Posee una versión libre conocida como UDK.
- No tiene soporte Web.
- Usado por grandes empresas para la creación de juegos.
- Ganador de varios premios.
- Posee herramientas de minería de datos.
- Gestor de contenido que permite ubicar los assets existentes.
- Manejo de luces sin la necesidad de herramientas externas.
- Permite la edición del código en C++ para realizar cambios.
- Soporta con DirectX y OpenGL.

4.2.3. CryEngine

CryEngine en un motor de videojuegos 2D y 3D que permite la creación de simulaciones y aplicaciones interactivas gracias a las amplias características que posee entre ellas el manejo de la aplicación en tiempo real, razón por la cual en el año 2010 obtuvo el premio a la mejor simulación en tiempo real.

CryEngine cuenta con un conjunto de herramientas avanzadas de animación que permiten crear personajes físicamente realistas, además gracias a sus componentes flexibles y escalables de Inteligencia Artificial, los personajes responden de manera inteligente a los cambios del juego y muestran comportamientos más realistas, generando sensación de comportamiento inteligente por parte de ellos. Este motor cuenta además con un procesador muy rápido por lo que se presenta una alta calidad de gráficos en tiempo real dando la impresión de fotorealismo tanto en ambientes internos como al aire libre.

Crytek es la empresa encargada de CryEngine que es una herramienta de software propietario, existe además de una versión completa que puede ser descargada y usada libremente mientras los productos que se desarrollen no tengan un fin comercial.

Este motor se encuentra disponible para Windows y las aplicaciones desarrolladas pueden correr en PlayStation 3, Xbox 360 y Sistema Operativo Windows. Una de las ventajas que ofrece Crytek en su motor es la posibilidad de desarrollar simultáneamente en un solo editor, mostrando así los resultados y cambios en tiempo real a cada uno de los desarrolladores lo que recaerá en una reducción significativa del riesgo en el desarrollo del juego.

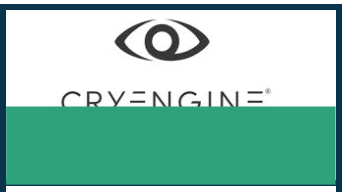
Características:

- Sistema operativo propietario.
- Disponible en versión gratuita para uso no comercial.
- Usa Microsoft Windows como sistema operativo para su programación.
- Multiplataforma, disponible para PS3, Xbox 360 y Windows.
- Manejo de un solo editor en tiempo real.
- Sistema de partículas en tiempo real.
- Manejo de inteligencia artificial.
- Procesador de alta velocidad.
- Iluminación dinámica en tiempo real.

4.2.4. Comparación de motores de video juegos

Una vez comparadas las herramientas de modelado y descritos los motores de video juego existentes, se puede realizar una comparación entre ellos para elegir cual es el mejor de acuerdo a las necesidades y que permita realizar el proyecto de la manera más eficiente; teniendo en cuenta aspectos básicos como la proyección que se le ha dado al proyecto y el sistema operativo instalado en los equipos a utilizarse en el desarrollo (Ver Tabla IV.II).

TABLA IV. IV. COMPARACIÓN DE MOTORES DE VIDEO JUEGOS

			
PRECIO	Unity es gratis para las versiones Indie de PC, Android y iOS, Existe una versión llamada Pro que cuesta $1500. Las licencias para Wii, Xbox 360, PS3 u otras consolas se venden por separado y cuestan dependiendo del proyecto y la cantidad de gente involucrada en este.	Unreal Development Kites gratis para proyectos nocomerciales. Si se quiere usar comercialmente cuesta $99 por cada computadora y si las ganancias pasan de $50000, Epic Games, se queda con el25% de las ganancias.	Gratis para proyectos en los que no se tenga ganancias, si es para indies se puede conseguir una licencia sin costo en la que CryTek se queda con el 20% de las ganancias; si es un proyecto más grande entonces hay que contactarse con Crytek para negociar.
FACILIDAD DE APRENDIZAJE Y USO	Unity es muy fácil e intuitivo para su aprendizaje y su uso. Su interfaz es muy simple y de fácil y rápida adaptación lo que permite usarlo con tan solo ver unos tutoriales de inicio. Unity cuenta con answers.unity3d. que es una comunida de de apoyo muy útil, exiten además varios sitios que se dedican a ayudar al proyecto o publicar tutoriales que se pueden encontrarlos por ejemplo en Youtube o ITunes.	UDK puede ser un poco difícil al inicio al menos para personas que no estén familiarizados con el mundo de Gráficos por Computadora, pero con el uso frecuente se vuelve fácil de usar. UDK tiene muy buenos tutoriales en la Web, pero estos no suelen ser gratis, un lugar en el que se los encuentra es eat3d.com. La documentación está muy bien escrita y tiene un foro propio.	CryEngine cuenta con muy poca información pública y gratuita, para conseguir información existen buenos tutoriales que deben comprarse. CryEngine gracias a su Sandbox permite la fácil creación de niveles en mundos abiertos, en espacios interiores se vuelve complejo

(Continuación) TABLA IV.II. COMPARACIÓN DE MOTORES DE VIDEO JUEGOS

GRÁFICOS	Los gráficos en Unity son de última generación y los desarrolladores siempre buscan mejorar. Para utilizar mejores gráficos tales como sombras en tiempo real y mapeo de luces con mapas de normales se debe comprar la licencia Pro.	Los gráficos en UDK son muy realistas. El sistema de luces es el más avanzado de todos los motores actuales. UDK permite renderizar dinámicamente mapas de luces e iluminación global en tiempo real.	Es el mejor para lograr renderizados espectaculares en tiempo real, con soporte para DirectX 11, su limitación es que no deja crear o modificar los shaders con los que viene por defecto
FISICA	Unity tiene una física realista, softbodies, rigidbodies, colliders y más componentes que se pueden unir fácilmente. También se puede añadir materiales físicos a un collider y ajustar la cantidad de fricción y rebote. Tiene un sistema de simulación de telas que está siendo mejorado. Fácilmente se puede añadir plugins desde el Asset Store para fracturing, pero esta característica no se incluye por defecto.	UDK tiene un muy buen sistema de física. Este utiliza actores para darles un sistema de física basado en su malla. La simulación de tela es muy buena y la fractura de mallas con física realista es mejor. UDK permite además escoger la complejidad del sistema de colisiones de cada objeto.	Posee un motor de física propio, este es multi-threaded, lo cual hace que el cálculo de colisiones y fuerzas funcione muy rápido.
SCRIPTING	Para escribir código se usa MonoDevelop que permite codificar en JavaScript (UnityScript), C#,o Boo. También hay soluciones visuales para scripting en el Asset Store, y muchas de estas son gratis.	En UDK, se puede hacer un scripteo básico con el sistema visual llamado Kismet. Para todo lo complejo se debe usar yScript y C++ pero esto solo se activa en la versión pagada.	Tiene soporte para Lua y C++, además de tener un sistema visual de scripteo.
PLATAFORMAS DE DESARROLLO	Permite el desarrollo en Windows y Mac	Soporta solamente Windows para el desarrollo	Soporta solamente Windows para el desarrollo
PLATAFORMAS SOPORTADAS	Unity3D puede publicar con un solo click para: Adobe Flash, iOS, Android, Windows, Mac, Linux, Windows Phone, Windows Store, Blackberry, Nintendo Wii, Nintendo Wii U, PS3, PS4, y XBOX 360; así como para la Web	UDK puede publicar para: Adobe Flash, iOS, XBOX 360, PS3, PS4, PS Vita, Windows, Android y Mac	Se puede exportar para PC, Xbox 360, Xbox One, PS3, PS4, Wii U, Android y iOS, para cada una de estas se necesita una licencia diferente.
EXPORTAR PARA LA WEB	Si	No	No
SPLASH SCREEN (Pantalla de Carga)	Unity permite cambiar el splash screen pero solo con en versión Pro, en la versión gratis aparece un splash screen de Unity.	UDK no permite cambiar el splash screen.	Se puede usar el splash screen que se desee.

(Continuación) TABLA IV.II. COMPARACION DE MOTORES DE VIDEO JUEGOS

IMPORTAR ARCHIVOS	Unity puede importar casi todos los formatos de archivos 3D, texturas, mapas, archivos de audio y video. El tipo de archivo FBX es el mejor para trabajar con animaciones en Unity. El proceso de importar un archivo es muy fácil, solamente se arrastra y suelta como si fuera un explorador normal del sistema en la pestaña, cuando se realizan cambios a los archivos importados con herramientas externas Unity los actualizará automáticamente	UDK no soporta la mayor parte de archivos comunes, y necesita que se importen en un paquete .upk. Los assetos no se actualizan automáticamente y tampoco es posible verlos desde algún explorador común	Hay que usar plugins especiales para los programas de diseño, modelado y animación para que exporten en formatos propios del CryEngine, lo cual limita bastante los programas que se pueda usar para la creación de modelos, animaciones y texturas
ADMINISTRACION DE ASSETS	Unity tiene una pestaña llamada Proyecto en el editor, que ayuda a tener todo organizado y funciona como cualquier explorador del sistema de los que se acostumbra a usar. Permite también el uso de etiquetas sobre los objetos para buscarlos y organizarlos más fácil.	UDK tiene un Buscador de Contenido que es muy eficiente y que permite una organización optima de assets a pesar de ser diferente a los usados en exploradores comunes	CryEngine cuenta con el Sandbox que posee un explorador para el manejo de assets
JUEGOS NOTABLES HECHOS CON EL MOTOR	BeGone, Cowboy Guns, Ghost Recon Network, Battlestar Galactica Online, Max and the Magic Marker, Red Crucible 2, Shadowgun, Uberstrike, Angry Birds Bad Piggies, Star Wars: The Quest for R2D2 y Temple Run: Brave.	Batman Arkham City, Borderlands 2, Mass Effect series, Gears of War series, DC Universe Online, Infinity Blade 1 y 2, Bioshock series, y UT3.	Crysis 3, Ryse, Son of Rome, Crysis, Crysis Warhead, Aion

Fuente: Propia

CAPITULO V

METODOLOGÍA

La siguiente fase es crear la metodología que debe ser independiente de las herramientas de hardware y software.

Para la creación del Mundo Virtual de la ESPOCH es necesaria la correcta integración de una herramienta de animación y modelado con un motor de video juegos; para cumplir con este requerimiento se pretende la creación de una metodología que esté de acuerdo al alcance que se pretenda dar al producto final, y a las condiciones con las que se cuenta; tratando así de lograr los mejores resultados visuales y funcionales del proyecto.

Para integrar una herramienta de animación y modelado con un motor 3D vamos a utilizar la metodología mostrada en la imagen. Esta metodología incluye la exportación de texturas, interfaces, modelos, animaciones y sonidos desde herramientas de diseño, modelado, animación o banco de sonidos respectivamente para su posterior integración y programación dentro del motor 3D generando como resultado un mundo virtual al cual se lo implantará y se realizaran las pruebas respectivas (Ver Figura 16).

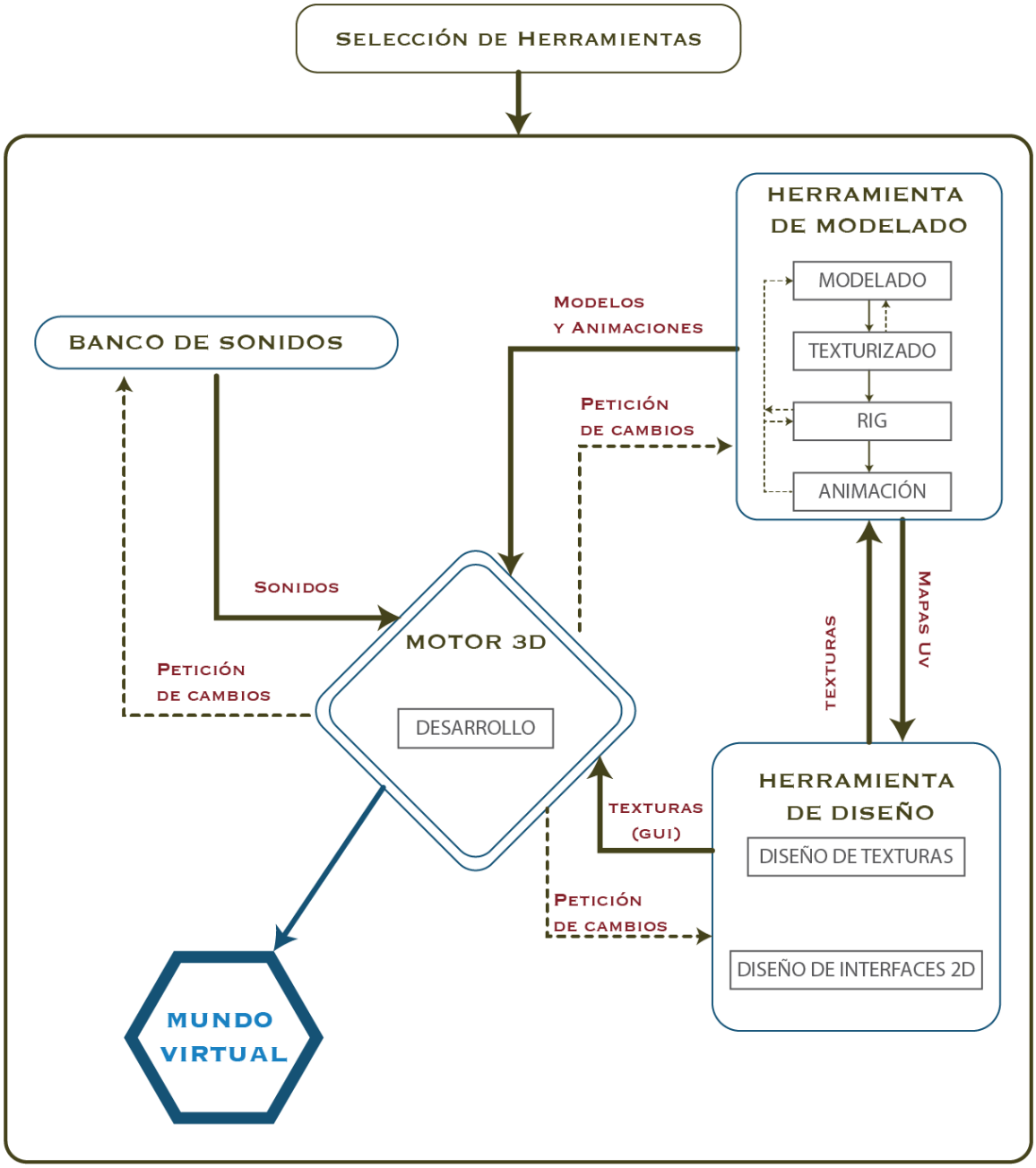


Figura 16. Metodología General

(Fuente: Propia)

5.1. Selección de herramientas

5.1.1. Herramienta de animación y modelado seleccionada

Tomando en cuenta las características principales de cada herramienta se ha llegado a la conclusión que para el presente estudio la mejor herramienta es Autodesk Maya 2014, ya que tanto el arte como la programación se va a hacer en computadoras que usan como Sistema Operativo Mac OSX 10.9, se puede descartar a Autodesk 3dsMax ya que solo funciona en Windows. Ahora si nos fijamos en el resto de las características como cantidad de polígonos que pueden manejar, manejo de UVs para texturizado y creación de animaciones, tanto Maya 2014, Cinema4d y Blender son capaces de hacer todo lo necesario para este proyecto, la diferencia viene dado en su facilidad tanto para nuevos usuarios como para usuarios expertos en su uso, y en esto es en lo que sobresale Autodesk Maya 2014 del resto de competidores, esto se confirma por la cantidad de empresas dedicadas a hacer videojuegos que lo usan.

5.1.2. Motor de video juegos seleccionado

Los 3 motores analizados son un estándar en la industria de los videojuegos, cada uno tiene sus beneficios y sus contras, normalmente se usa el CryEngine para hacer juegos en los que se pueda apreciar con mucho detalle escenas fuera de edificios, selvas o lugares abiertos, mientras que el UDK funciona mucho mejor para escenas cerradas, dentro de edificios en donde la visión este limitada por paredes.

Unity es uno de los motores más nuevos existentes en el mercado y ofrece a sus usuarios los aspectos más importantes que pueden ser encontrados en motores más antiguos y conocidos; es mucho más fácil en su manejo gracias a la interfaz que permite arrastras y pegar los assets en el proyecto.

Unity 3D es considerada una herramienta intuitiva y de fácil aprendizaje cuando alguien está empezando en el área del desarrollo de videojuegos, lo que

permite que el aprendizaje sea rápido y lleve muy poco tiempo el desarrollo completo de un juego.
Si bien se puede hacer casi cualquier tipo de juego con estos motores, su uso más común son los FPS(First Person Shooters); Unity 3D ha empleado su tiempo en el mercado para optimizar su trabajo con dispositivos móviles que tengan poco poder de procesamiento en relación a las computadoras y consolas de videojuegos de última generación.
Comparando el manejo de procesos de renderizado complejos, en la actualidad, tanto UDK como CryEngine presentan una ventaja visual sobre Unity 3D en el desarrollo de proyectos.

Debe tomarse en cuenta que el proyecto planteado deberá ser accesible desde la Web lo cual solo es factible al realizarlo en Unity 3D, ya que ninguno de los otros motores permite el desarrollo para Web y publicar la aplicación en un servidor para el acceso desde un navegador.

5.2. Metodología del desarrollo

Una vez que han sido seleccionadas las herramientas a utilizarse en la demostración de la metodología, se procederá a diagramar y ejecutar la metodología genérica para el desarrollo de Mundos Virtuales.

En la figura 17 se ve el diagrama de las fases específicas que serán implementadas posteriormente.

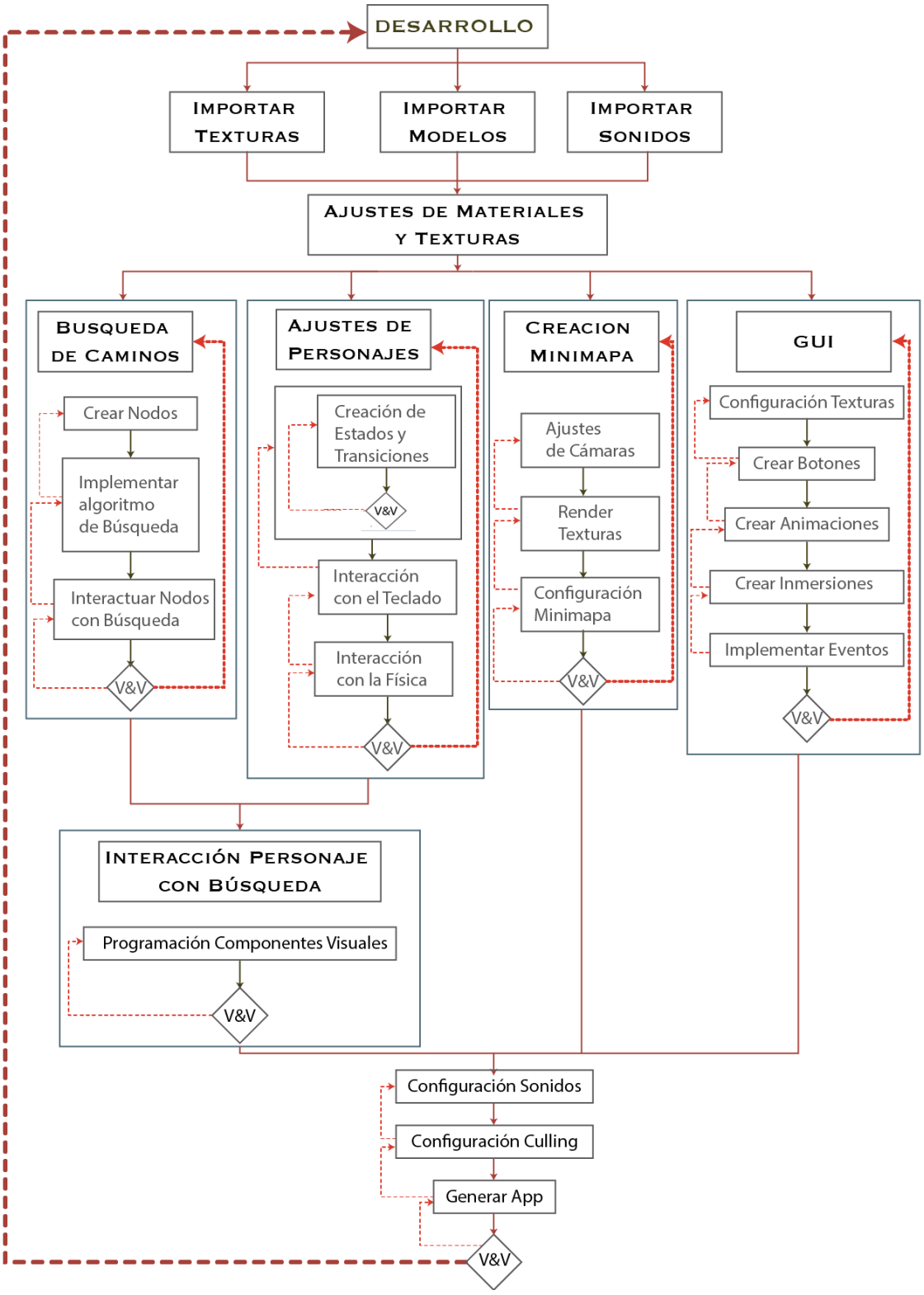


Figura 17. Metodología del desarrollo.

(Fuente: Propia)

5.3. Implementación de la metodología en las herramientas seleccionadas

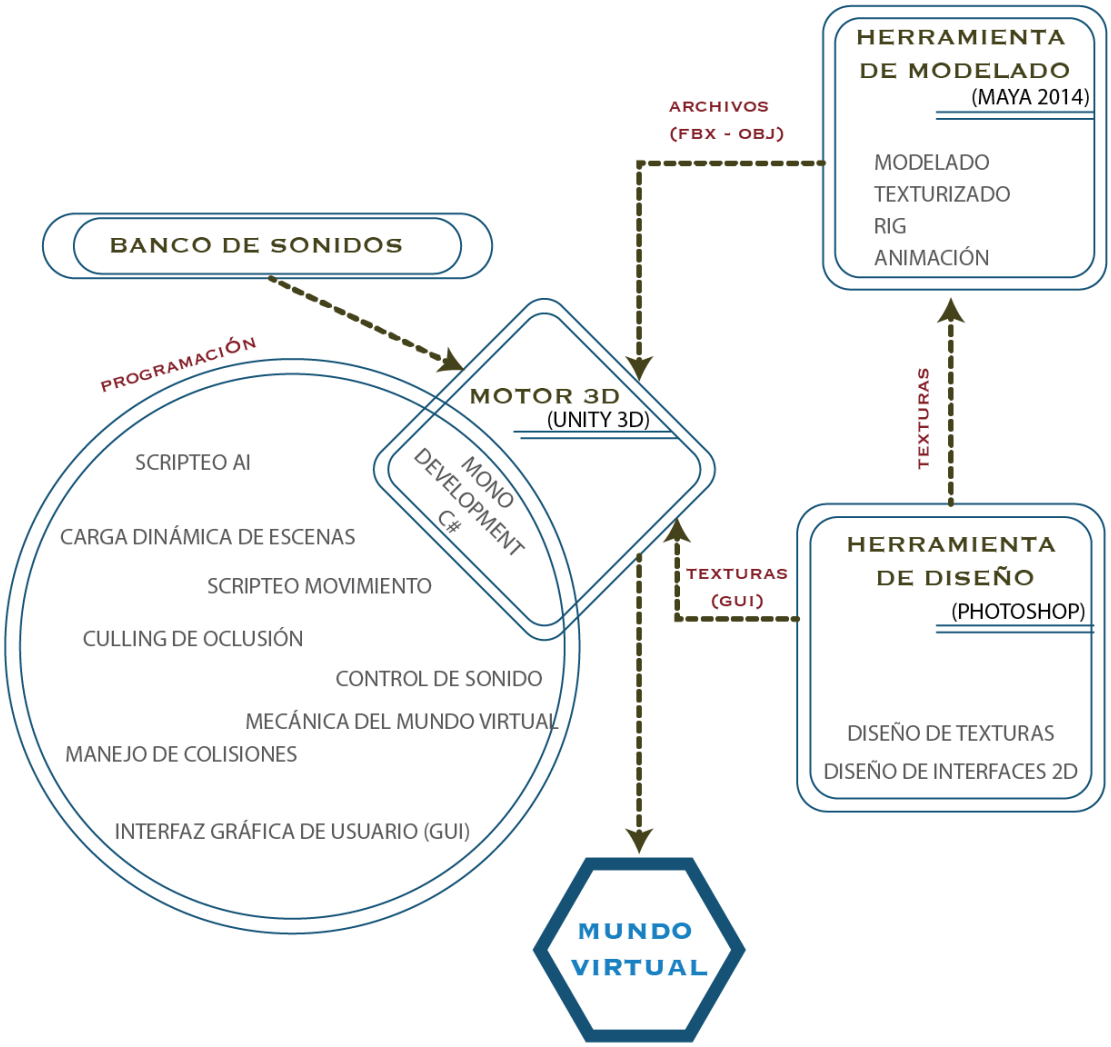


Figura 18. Modelo de la Metodología
(Fuente: Propia)

Por el estudio anterior se conoce que una de las ventajas de Unity es la facilidad que presenta para importar assets desde cualquier lugar del equipo hacia un directorio del proyecto, estos assets pueden ser importados desde cualquier herramienta de modelado sin importar con la extensión del archivo.

Los modelos son importados a Unity por medio de lo que se conoce como FBX importer; esta herramienta permite que el motor cambie internamente los archivos a extensión fbx desde cualquiera de las extensiones de las herramientas de modelado.

5.3.1. Creación de un nuevo proyecto

1. Buscar el ícono de Unity 3D en el escritorio y dar para empezar.

Figura 19. Icono de Unity en el Escritorio

2. Ir a la pestaña de *File* y seleccionamos *New* Project.

Figura 20. Crear un nuevo Proyecto

3. Escoger el directorio en el cual se quiere guardar el proyecto y clic en *Create Project.*

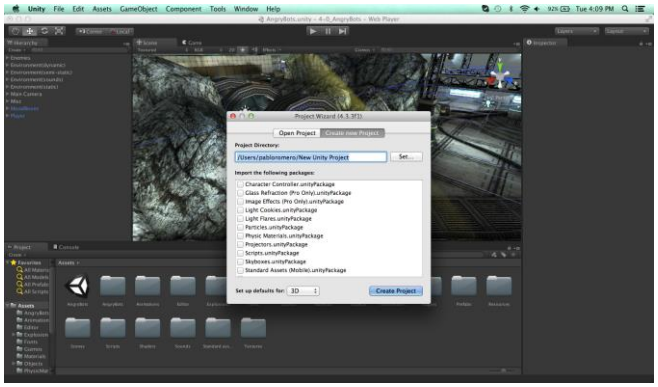


Figura 21. Guardar el proyecto

4. Se mostrará un proyecto vacío en el cual se trabajará.

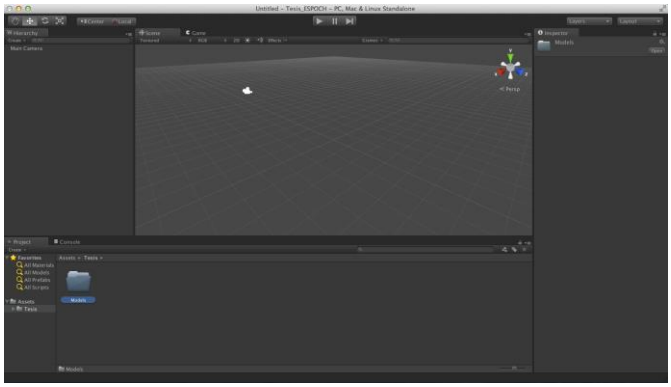
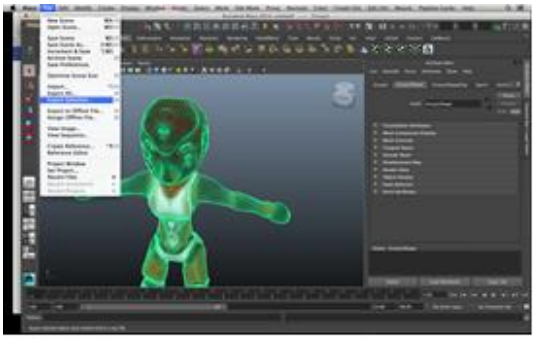


Figura 22. Proyecto vacío en Unity

5.3.2.Importar el modelo

1. Exportar el archivo en formato fbx desde Autodesk Maya.



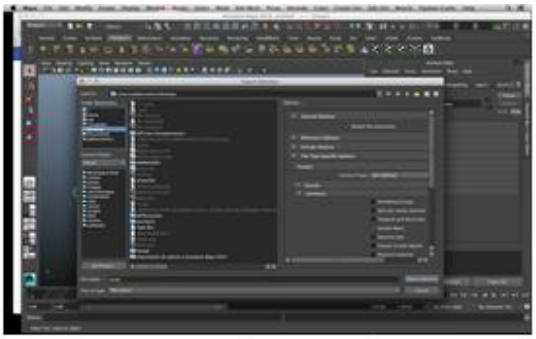


Figura 23. Exportar modelo desde Autodesk Maya

2. Buscar en el explorador el modelo exportado y arrastrar el archivo fbx del modelo a utilizar en la carpeta “Model” en el subdirectorio “Tesis” ubicado dentro del directorio “Assets” del proyecto de Unity.

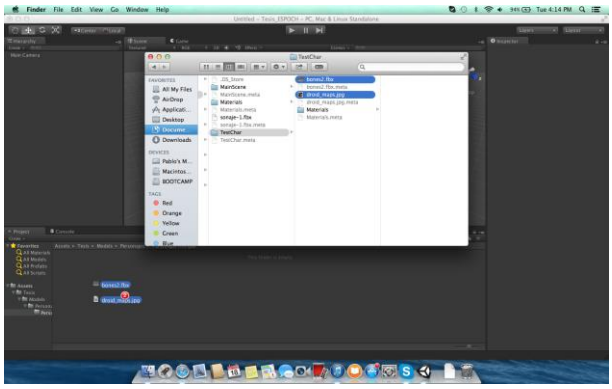


Figura 24. Buscar el modelo exportado

3. Clic en la Vista del Proyecto para mostrar el *FBX Importer* en el *Inspector*, esta es la interfaz utilizada para controlar como se importa el modelo hacia el proyecto en Unity.

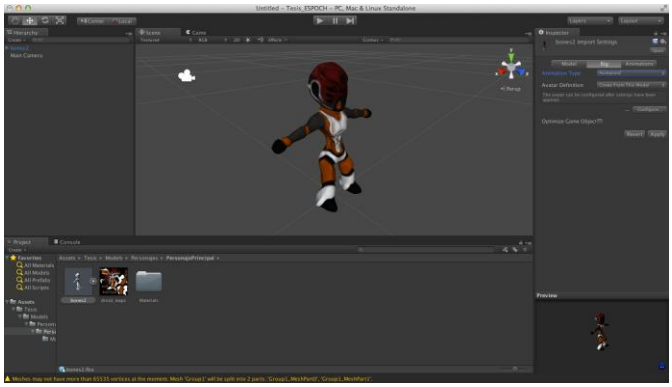


Figura 25. FBX Importer

4. Los modelos son importados en Unity por defecto usando una escala de 0.01; este valor puede ser cambiado en la opción Factor de Escala del *FBX importer*, para ajustarlo correctamente según lo que se necesita, mover el modelo a la Vista de Escena cerca del Controlador de Primera Persona y ejecutar el proyecto para encontrar la escala adecuada; una vez que se tenga el valor deseado se lo ingresar y presionar *Aplicar* para que se guarden los cambios.

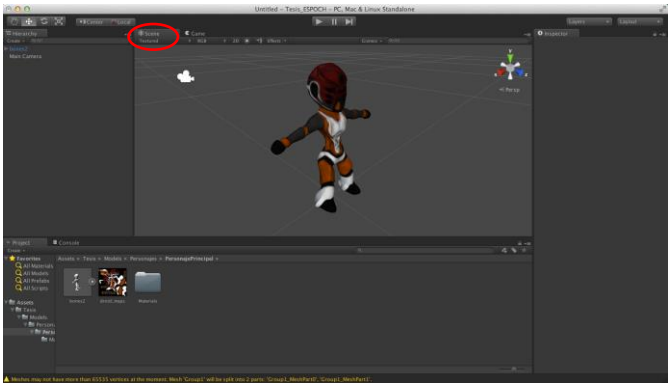


Figura 26. Buscar la escala del modelo

5.3.3.Importar texturas

Para importar una textura se realizan los siguientes pasos:

1. En la *Vista de Proyecto* se tiene por defecto una carpeta inicial llamada “Assets”, dentro de ella crear una nueva carpeta llamada “Texturas” en la carpeta que se llamó previamente “Tesis”.

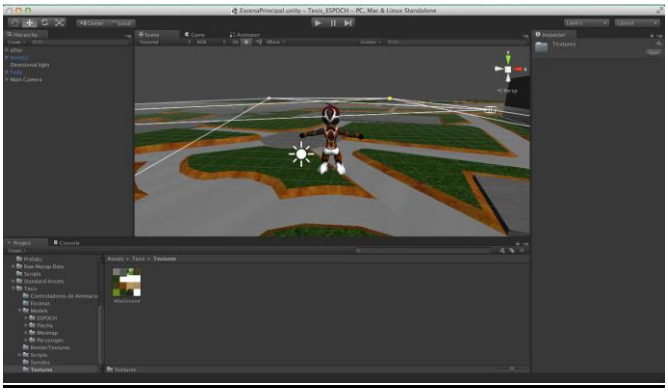


Figura 27. Carpeta Texturas

2. Buscar en el explorador la textura del objeto que se encuentra en formato .jpg, aunque podría estar en cualquiera de los formatos aceptados por Unity como png, psd, tif, entre otros.

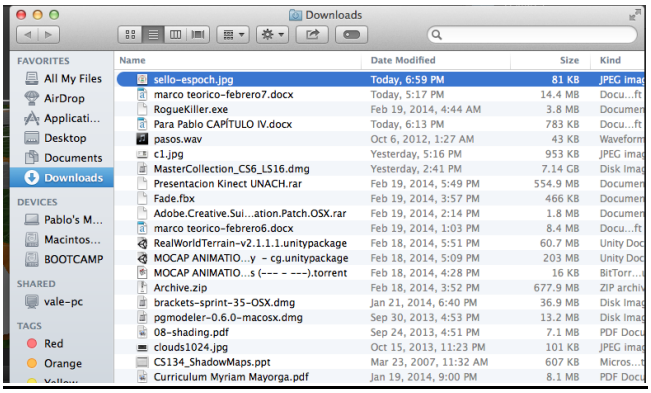


Figura 28. Buscar la textura en el explorador

3. Seleccionar la textura y la arrastrarla hasta Unity para pegarla en la carpeta que se creó en el paso anterior.

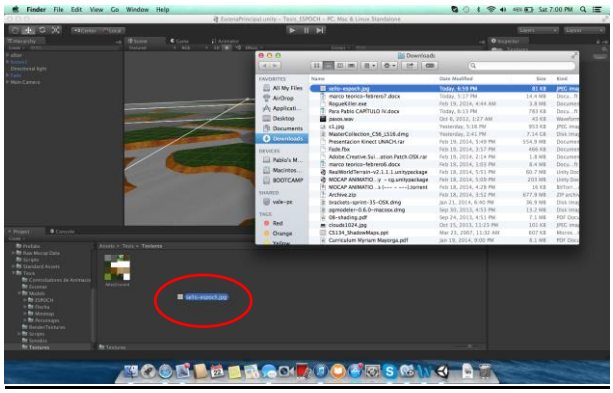


Figura 29. Arrastrar la textura hacia Unity

4. La textura será importada de manera correcta e internamente Unity la transformará al formato compatible necesario por medio de la creación de metadata.

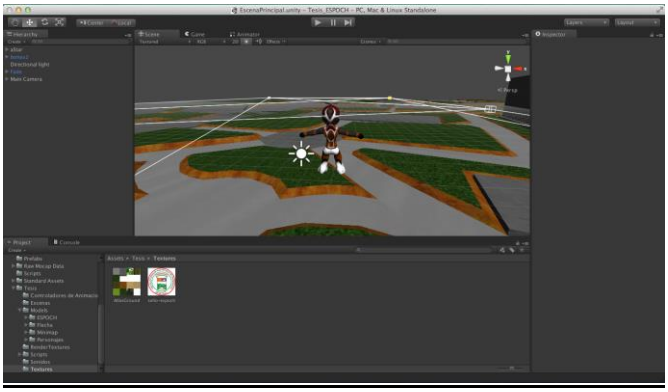


Figura 30. Textura Importada en Unity

5.3.4.Importar sonidos

Para obtener sonidos se puede crearlos u obtenerlos de una página Web que contenga bancos de sonidos.

Si los sonidos han sido creados o descargados desde alguna página Web deberán ser guardados en el explorador.

Para importarlos al proyecto se realizan los siguientes pasos:

1. En la Vista de Proyecto se tiene por defecto una carpeta inicial llamada “Assets”, dentro de ella crear una nueva carpeta llamada “Sonidos” en la carpeta que se llamo previamente “Tesis”.

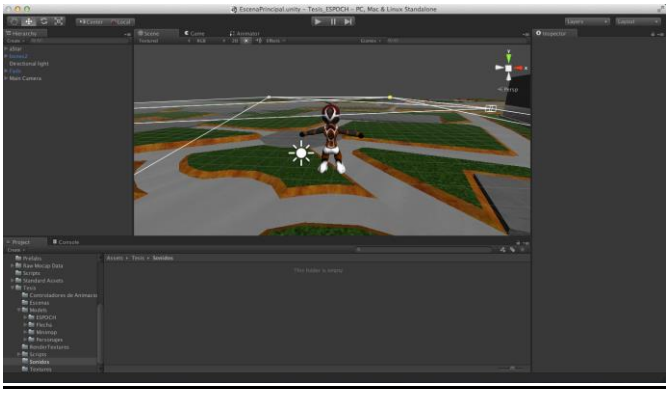


Figura 31. Carpeta Sonidos

2. Buscar en el explorador el banco de sonidos que se necesita importar al proyecto, estos sonidos pueden estar en aif, wav, mp3 y ogg entre otros.

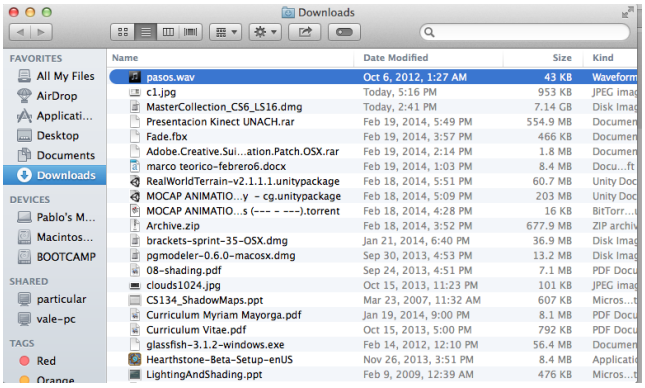


Figura 32. Buscar sonidos en el explorador

3. Seleccionar el banco de sonidos necesario y arrastrarlo hasta la carpeta previamente creada donde se lo pegará.

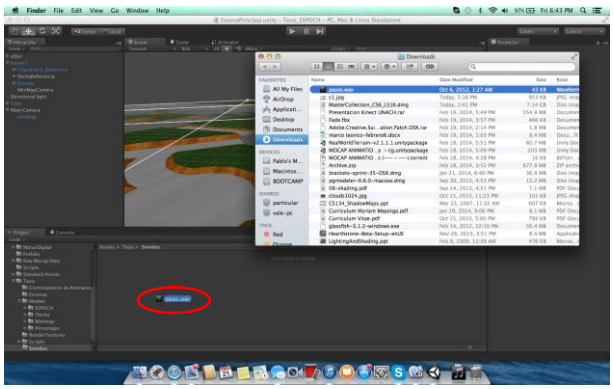


Figura 33. Arrastrar los sonidos a Unity

4. El banco de sonidos se importará de manera correcta e internamente Unity lo transformará al formato compatible necesario por medio de la creación de metadata.

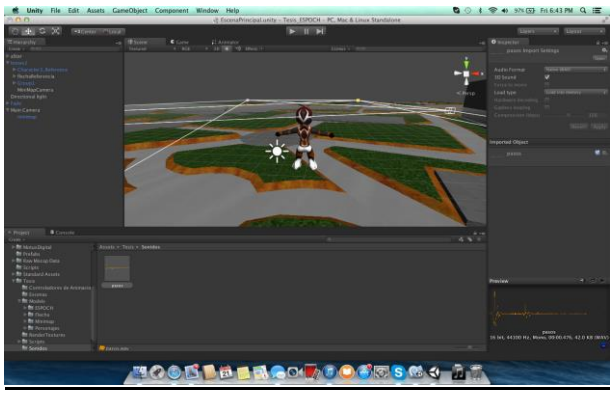


Figura 34. Sonido importado en Unity

5.3.5.Ajuste de materiales y texturas

Al importarse los modelos pueden presentarse errores en los materiales y texturas para lo cual se deben realizar ajustes en las propiedades de los mismos.

1. En la Vista de Jerarquía buscar el *modelo* y dar un clic sobre él para desplegar su jerarquía y sus opciones de configuración en el *Inspector*.

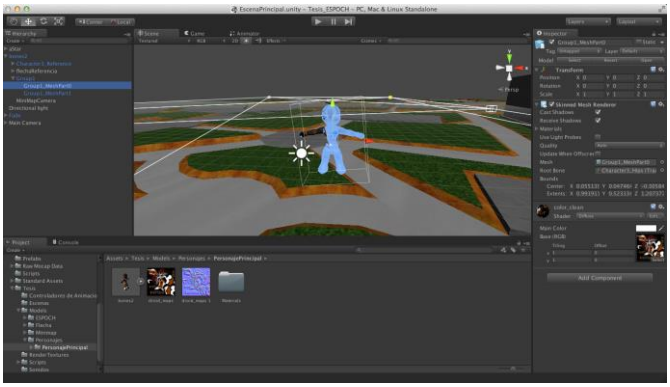


Figura 35. Modelo en Unity

2. Seleccionar el *mesh* del modelo, el cual tendrá un material de tipo “Diffuse”. En el Inspector seleccionar en *shader*: “Bumped Diffuse”.

Figura 36. Cambiar el shader al modelo

3. Seleccionar los colores. *Main Color*: es el color principal del objeto y se lo deja con el valor por defecto; *Base* (*RGB*): seleccionar la textura del objeto desde la carpeta donde se encuentre el proyecto en la *Vista* de *Proyecto*.

Figura 37. Agregar una Base(RGB) al modelo

4. *Normalmap*: seleccionar el mapa de normales desde la carpeta del proyecto de la misma manera que se hizo con *Base*(*RGB*).

Figura 38. Agregar un mapa de normales al modelo

5. Si aparece un mensaje sobre el control del mapa de normales indicando que no ha sido aún designado como un mapa de normales, presionar *Fix* y Unity arreglará el problema internamente.

5.3.6.Obtención de animaciones

Se pueden obtener animaciones gratuitas desde Unity por medio del Asset Store, para esto se realizan los siguientes pasos:

1. Abrir la pestaña *Window* y seleccionar *Asset* Store.

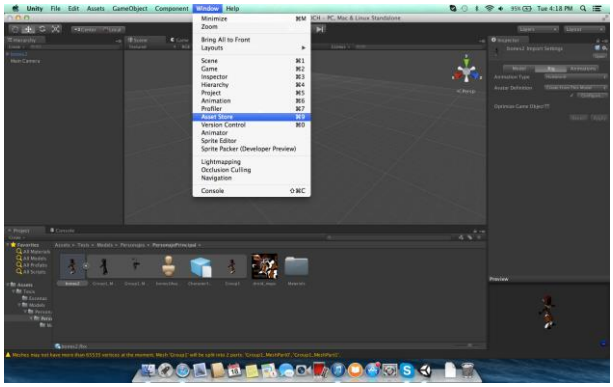


Figura 39. Asset Store desde Unity

2. Se despliega una ventana en la cual se selecciona *Animation* del menú *Categories* ubicado en la parte superior derecha.

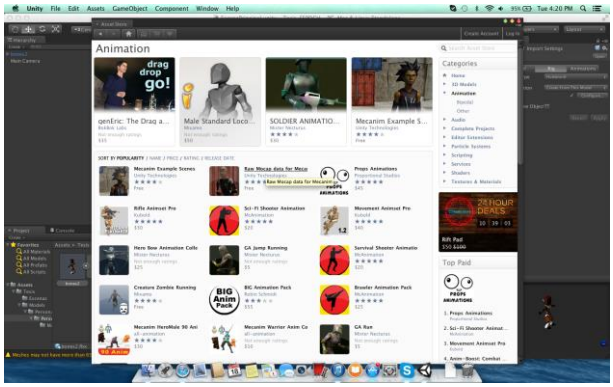


Figura 40. Menu Principal de Asset Store

3. Seleccionar la opción Raw Mocap data for Mecanim ubicada Panel Principal y dar clic sobre ella.

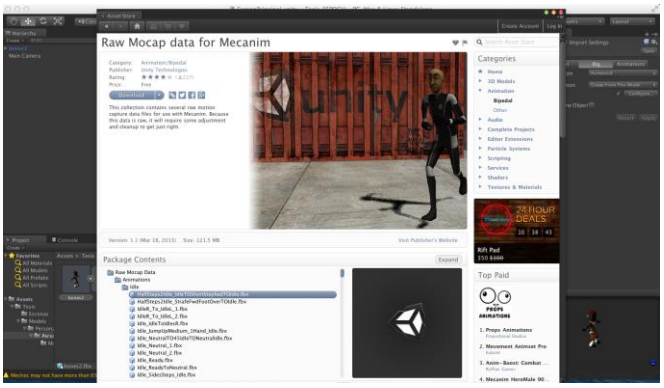


Figura 41. Raw Mocap data for Mecanim

4. Se muestra la descripción y ciertas características de la opción seleccionada y se muestra un botón que permite la descarga llamado *Download* sobre el cual se debe dar clic. Para iniciar la descarga se pedirá ingresar una cuenta o la creación de una cuenta.

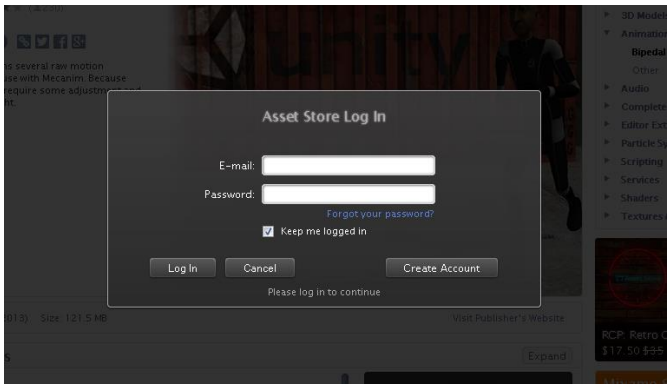


Figura 42. Descargar desde el Asset Store

5. Ingresar los datos personales que son requeridos para la creación de la cuenta, aceptar los términos de uso y se pedirá que se confirme la creación de la cuenta por medio del correo electrónico ingresado.

Figura 43. Crear cuenta en Asset Store para descargas

6. Una vez finalizada la descarga se muestra un menú que permite la importación de las animaciones al proyecto. Dar clic en *Import*.

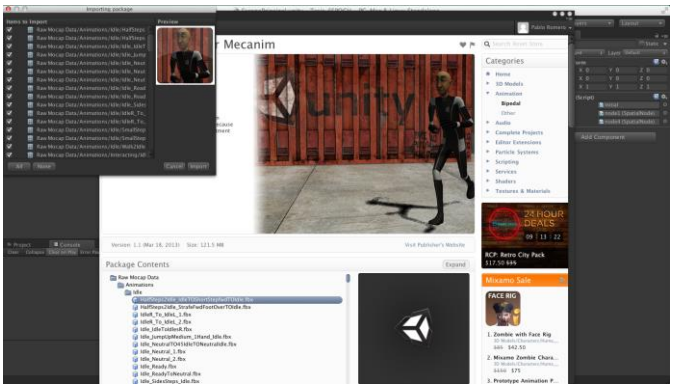


Figura 44. Importar animaciones a Unity

5.3.7.Ajuste de animaciones

1. Para ajustar las animaciones, seleccionar el *modelo* en fbx en la Vista de Proyecto y en la Vista de Inspector seleccionar *Rig*.

Figura 45. Modelo previo a la animación

2. En *Animation Type* seleccionar *Humanoid* y clic en el botón *Configure*.

Figura 46. Configuración de Animation Type

3. Si están correctamente configurados los huesos del Avatar, se mostrarán todos en verde, caso contario arrastrarlos desde la V*ista de Jerarquía* hacia el *Inspector*. Una vez terminado clic en *Done*.

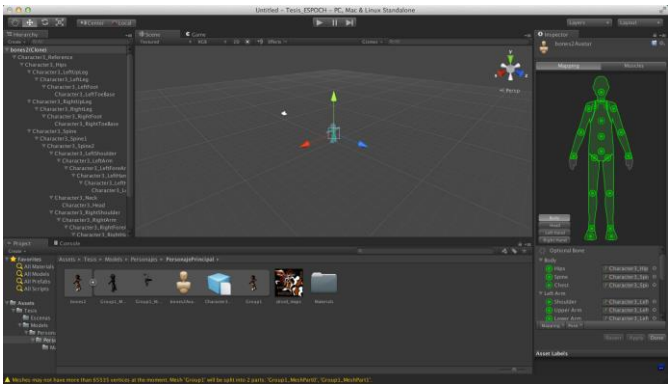


Figura 47. Configuración de huesos del modelo

4. Seleccionar la pestaña *Animations* ubicada en la parte superior del *Inspector*. En el *List Panel* de las animaciones clic en "**+**" que se encuentra en el lado derecho y se agregará una nueva animación.

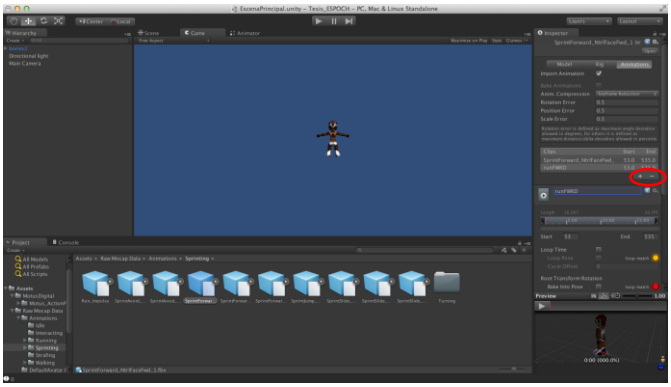


Figura 48. Agregar una nueva animación

5. En *Start* dejar como valor de inicio, por ejemplo, 91 y en *End* se pondrá el número de frame en el que terminará la animación por ejemplo 115.

   Si se desea que la animación se vuelva a reproducir una vez acabada marcar la casilla de *Loop Time*, se debe tener en cuenta que la animación sea coherente en su primer y último frame, esto se comprueba si los cuatro botones de *loop match* están de color verde.

   Clic en el botón de *Aplicar* para guardar los cambios.

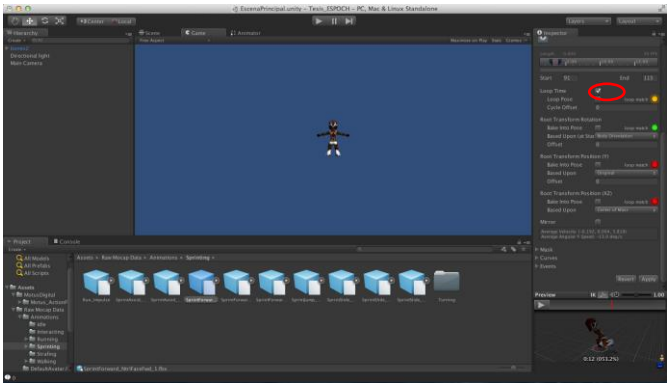


Figura 49. Configurar valores de la animación

5.3.8.Creación de transiciones y estados para la animación

1. En la Vista de Proyecto crear un nuevo Animator Controller dentro del cual se configurarán las transiciones entre las animaciones.

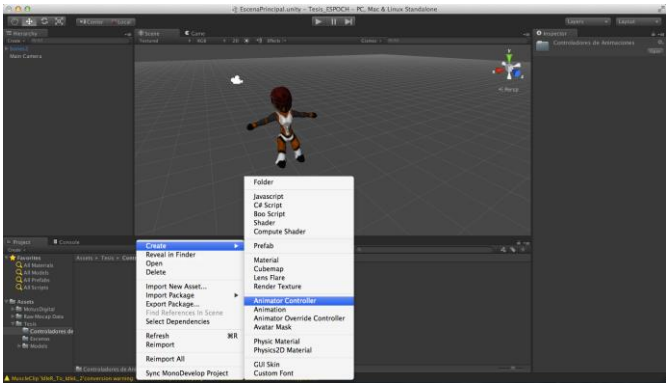


Figura 50. Crear un Animatior Controller

2. Para crear un nuevo estado, dar clic derecho en la *Vista de Proyecto* Create State y seleccionar *Empty*.

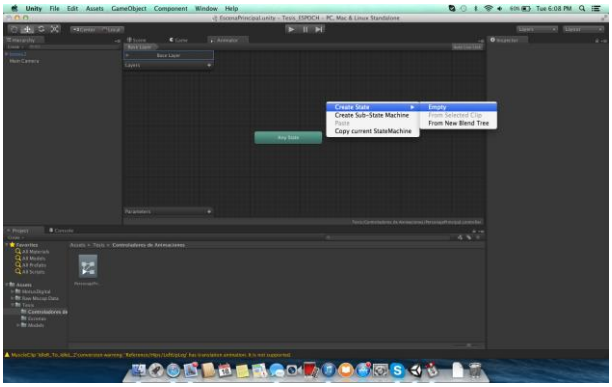


Figura 51. Crear un nuevo estado

3. Clic en el Estado creado y en el Inspector ir a *Motion*.
   Clic en el botón de la derecha y seleccionar la animación que se va a agregar y aceptar con doble clic en ella.

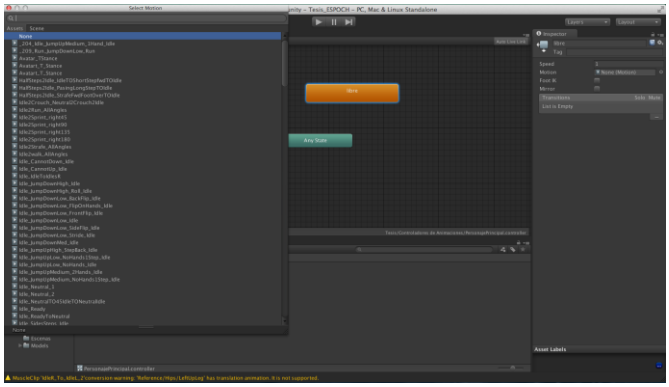


Figura 52. Agregar una animación a un estado

4. Crear un Segundo Estado y asignarle otra animación de la misma manera que con el estado anterior.

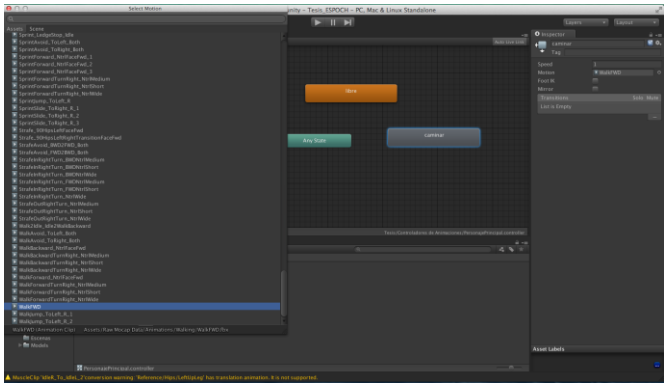


Figura 53. Agregar una animación al segundo estado

5. Para agregar una Transición clic derecho sobre el Primer Estado y seleccionar *Make Transition*.

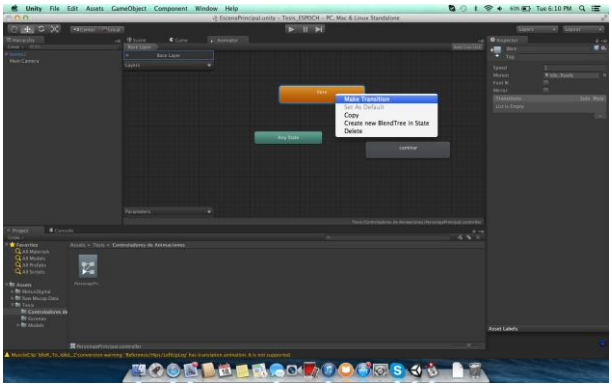


Figura 54. Agregar transición entre los estados

6. Clic sobre el Estado al que se quiere llegar (Segundo Estado). En la parte inferior izquierda del Animator clic en "**+**" junto a *Parameters* y seleccionar la opción *Float*;

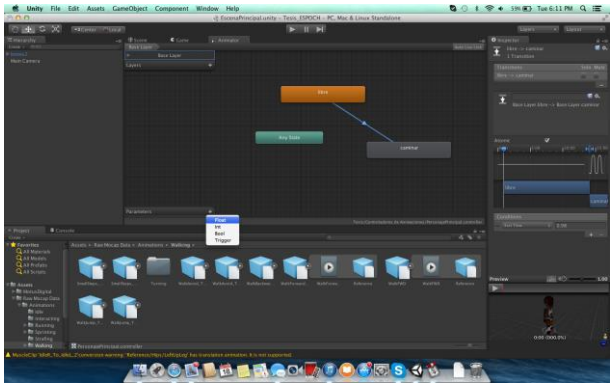


Figura 55. Crear nuevo parámetro

7. Como resultado se obtendrá una nueva variable a la que se la nombrará como *velocidad* que permitirá controlar las transiciones de movimiento.

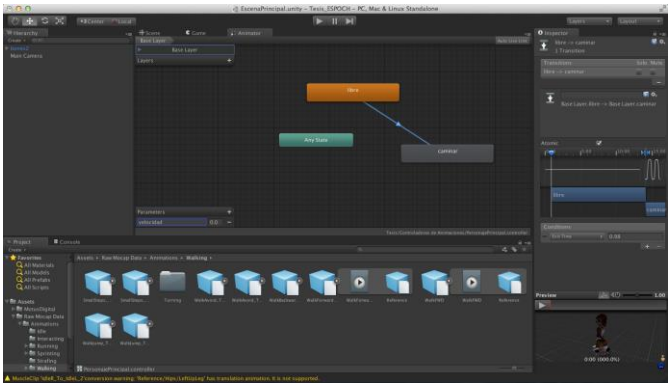


Figura 56. Nombrar la variable para controlar transiciones

8. Una vez creada la variable *velocidad* clic en la transición creada. En la parte inferior del *Inspector* ubicar *Conditions* en donde se seleccionará la variable recién creada.

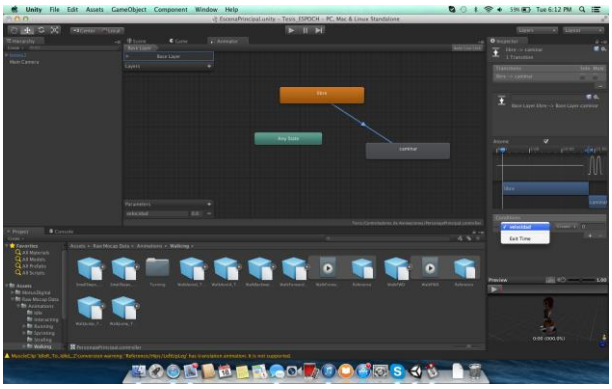


Figura 57. Agregar la variable creada a Conditions

9. Junto a la variable se encuentra un pick box que permitirá seleccionar si la comparación para realizar la transición se realizará por mayor o menor valor al que se lo asigna en el cuadro junto a él. Por ejemplo para pasar

de un estado de “parado” a “caminar” deberá superar una *velocidad* de 0.1, y de “caminar” a “correr” deberá superar una *velocidad* de 1.

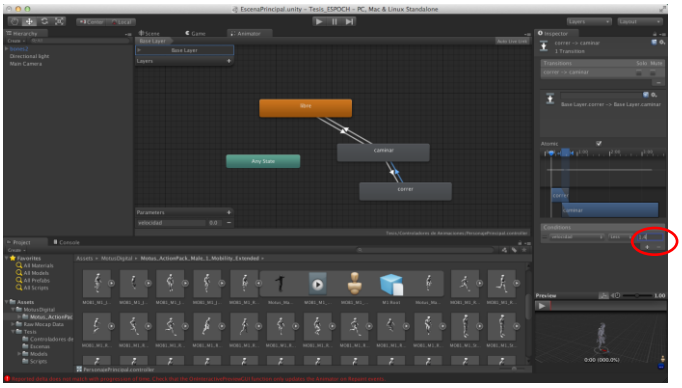


Figura 58. Asignar parámetros para ejecutar la transición

5.3.9. Probar las transiciones

Una vez creadas las transiciones y escenas de la animación agregarlas al modelo siguiendo los pasos descritos a continuación:

1. En la Vista de Proyecto buscar el *modelo* que se importó anteriormente, seleccionarlo y arrastrarlo hacia la Vista de Jerarquía.
   En la Vista de Jerarquía seleccionar el *modelo* que se agregó en el paso anterior y en el Inspector buscar el componente *Animator* del modelo; seleccionar el *Animator Controller* que se creó anteriormente en el campo mencionado.

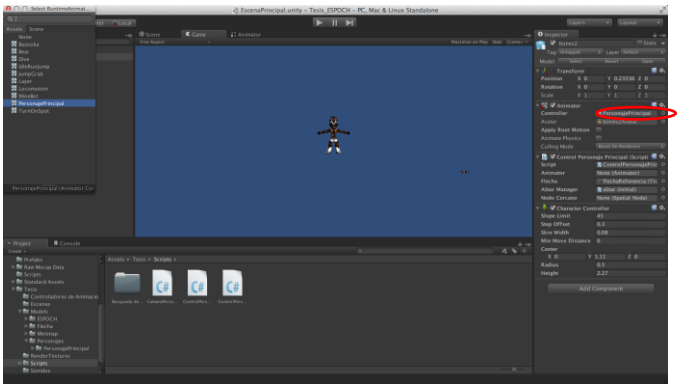


Figura 59. Agregar la animación al modelo

2. En la *Vista de Proyecto* crear una carpeta con el nombre “Scripts” y dentro de ella clic derecho y crear un nuevo componente C# Script al que se lo nombrará *ControlPersonajePrincipal*

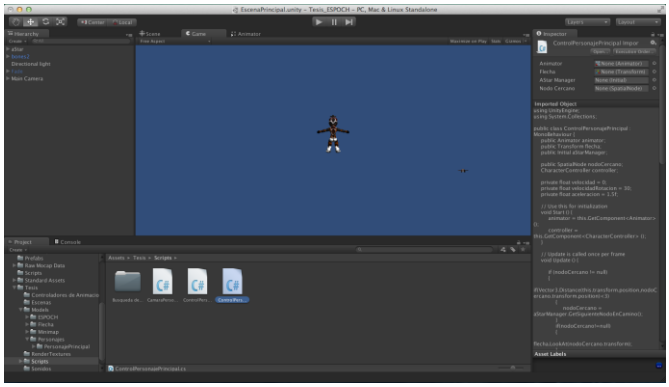


Figura 60. Crear el Script ControlPersonajePrincipal

3. Doble clic al elemento y se abrirá una nueva ventana con MonoDevelop que es el IDE que permite programar en Unity.

   El Script por defecto tendrá los procedimientos *Start y Update*. Start es llamado una única vez cuando se cree el objeto al que se le asignará este Script; mientras que Update se llamará cada vez que transcurra un frame en la aplicación.

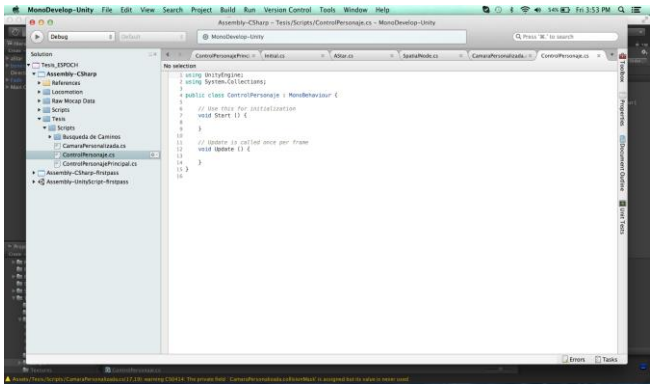


Figura 61. Script por defecto en Mono Develop

4. Crear una variable de tipo Animator a la que se la llamará *animator*, dentro del procedimiento Start a esta variable le asignamos los valores del componente Animator del modelo sobre el que se está trabajando. Dentro de este componente se encuentra la variable de la transición creada anteriormente, la cual tiene por nombre *velocidad* y a la que se le asignará el valor de 1.5 lo que permitirá que el modelo camine por cumplir las condiciones antes descritas.

```
using UnityEngine;
using System.Collections;

public class ControlPersonaje : MonoBehaviour {
    private Animator animator;
    // Use this for initialization
    void Start () {
        animator = this.GetComponent<Animator> ();
        animator.SetFloat ("velocidad", 1.5f);
    }

    // Update is called once per frame
    void Update () {

    }
}
```

Figura 62. Código para asignar un valor a la variable velocidad

5. Guardar los cambios y volver a la ventana de Unity donde se buscará la Vista de Jerarquía y se seleccionará el modelo. En el Inspector dar clic en *Add Component*, y buscar Scripts para seleccionar *ControlPersonajePrincipal* que es el Script recién creado.

Figura 63. Añadir el Script de animación al modelo

6. Para probar el funcionamiento del modelo recién programado dar clic en ► en la Vista la Escena lo que mostrará el personaje realizando la animación de caminar, cumpliendo con lo que se asignó en la Creación de Transiciones que decía que si *velocidad* es mayor que 0.1 se pasa de la transición “parado” a “caminar”.

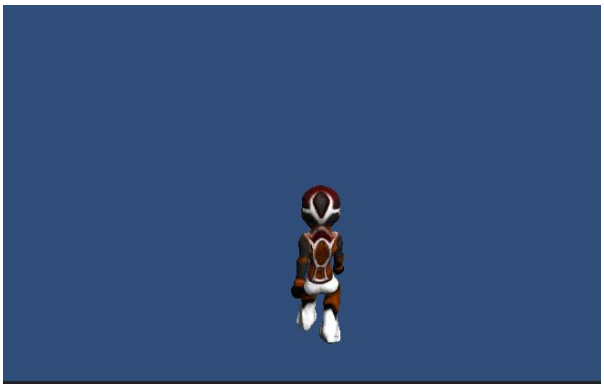


Figura 64. Prueba de funcionamiento de la animación

5.3.10.Añadir a la animación interacción con el teclado

Una vez que se ha probado que funcionan las transiciones se insertará una línea de código que permitirá que la animación sea controlada por el teclado.

1. Volver a MonoDevelop y borrar la línea de código que se creó anteriormente en la cual se la asignaba un valor a la velocidad

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class ControlPersonaje : MonoBehaviour {
5     private Animator animator;
6     // Use this for initialization
7     void Start () {
8         animator = this.GetComponent<Animator> ();
9         animator.SetFloat ("velocidad", 1.5f);
10    }
11
12    // Update is called once per frame
13    void Update () {
14
15    }
16 }
```

Figura 65. Código para asignar valor a la velocidad

2. Crear las variables *velocidad*, *velocidadRotacion*, *aceleracion*, de tipo float las cuales permitirán que se lleven a cabo las transiciones de la animación.

   Asignar a la variable *velocidad* el valor de 0, a la variable *velocidadRotacion* el valor de 30 y a la variable *aceleracion* 1.5f (se pone f al final del número para decir que es flotante) que serán los valores de inicio.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class ControlPersonaje : MonoBehaviour {
5     private Animator animator;
6
7     private float velocidad = 0;
8     private float velocidadRotacion = 30;
9     private float aceleracion = 1.5f;
10    // Use this for initialization
11    void Start () {
12        animator = this.GetComponent<Animator> ();
13
14    }
15
16    // Update is called once per frame
17    void Update () {
18
19    }
20 }
21
```

Figura 66. Código para creación de variables para interactuar con el teclado

3. Una vez que se ha asignado el valor inicial a las variables se las programará dentro de la función Update que se actualiza en cada frame.
   Para capturar el valor que envían las flechas de arriba ↑ y abajo↓ del teclado al ser presionadas se utiliza la variable *verticalAxis* a la cual se le asigna el valor que devuelve la función Input.GetAxis("Vertical") lo que devolverá valores entre -1 y 1 dependiendo de qué flecha este presionando y la cantidad de tiempo que se ha presionado.

```
16      // Update is called once per frame
17      void Update () {
18          float verticalAxis = Input.GetAxis ("Vertical");
```

Figura 67. Código para capturar la entrada del teclado

4. Una vez capturada la entrada del teclado se incrementará o disminuirá la velocidad que se le aplicará al modelo. Para esto a la variable *velocidad* se le asignará el valor de la *aceleracion* y se le multiplica por *Time*.deltaTime que devolverá el valor del tiempo que ha pasado desde el frame anterior en segundos. En cada frame se le sumará o restará nuevamente este valor dependiendo del sentido de la flecha que se presione en el teclado.

```
16      // Update is called once per frame
17      void Update () {
18          float verticalAxis = Input.GetAxis ("Vertical");
19          if (verticalAxis > 0)
20          {
21              velocidad += aceleracion * Time.deltaTime;
22          }
23          else if (verticalAxis < 0)
24          {
25              velocidad -= aceleracion * Time.deltaTime;
26          }
```

Figura 68. Código para cambiar el valor de la velocidad

5. Si se ha dejado de presionar una flecha en el teclado, para que la animación luzca real se dividirá el valor de la *aceleracion* entre 2 y se le disminuirá este valor a la *velocidad* hasta que ésta llegue a 0.

```
16      // Update is called once per frame
17      void Update () {
18          float verticalAxis = Input.GetAxis ("Vertical");
19          if (verticalAxis > 0)
20          {
21              velocidad += aceleracion * Time.deltaTime;
22          }
23          else if (verticalAxis < 0)
24          {
25              velocidad -= aceleracion * Time.deltaTime;
26          }
27          else
28          {
29              if (velocidad > 0.15f)
30              {
31                  velocidad -= aceleracion / 2 * Time.deltaTime;
32              }
33              else if(velocidad<-0.15f)
34              {
35                  velocidad += aceleracion / 2 * Time.deltaTime;
36              }
37              else
38              {
39                  velocidad = 0;
40              }
41          }
```

Figura 69. Código para programar la desaceleración

6. Para mantener la animación a una velocidad estándar se le asignará a la variable *velocidad,* los valores límite de -1 y 3 con la función Mathf.Clamp, la que permite que los valores entre el límite se asignen normalmente, caso contrario se tomará -1 si es un valor inferior o 3 si es un valor superior a los límites el resultado del cálculo de la *velocidad*.

```
16      // Update is called once per frame
17      void Update () {
18          float verticalAxis = Input.GetAxis ("Vertical");
19          if (verticalAxis > 0)
20          {
21              velocidad += aceleracion * Time.deltaTime;
22          }
23          else if (verticalAxis < 0)
24          {
25              velocidad -= aceleracion * Time.deltaTime;
26          }
27          else
28          {
29              if (velocidad > 0.15f)
30              {
31                  velocidad -= aceleracion / 2 * Time.deltaTime;
32              }
33              else if(velocidad<-0.15f)
34              {
35                  velocidad += aceleracion / 2 * Time.deltaTime;
36              }
37              else
38              {
39                  velocidad = 0;
40              }
41          }
42
43          velocidad = Mathf.Clamp (velocidad, -1, 3);
```

Figura 70. Código para truncar los valores obtenidos por el teclado

7. Para permitir la rotación del modelo se presionará las flechas izquierda ← o derecha → del teclado; la rotación se calcula en el eje Y del valor del ingreso del teclado y se le multiplica por la *velocidadRotacion* y por el Time.deltaTime.

   Esta rotación solo se realizará si el modelo está en movimiento lo cual se comprueba si la *velocidad* es mayor que 0.1 cuando camina para adelante o menor que -0,1 cuando camina para atrás.

```
16    // Update is called once per frame
17    void Update () {
18        float verticalAxis = Input.GetAxis ("Vertical");
19        if (verticalAxis > 0)
20        {
21            velocidad += aceleracion * Time.deltaTime;
22        }
23        else if (verticalAxis < 0)
24        {
25            velocidad -= aceleracion * Time.deltaTime;
26        }
27        else
28        {
29            if (velocidad > 0.15f)
30            {
31                velocidad -= aceleracion / 2 * Time.deltaTime;
32            }
33            else if(velocidad<-0.15f)
34            {
35                velocidad += aceleracion / 2 * Time.deltaTime;
36            }
37            else
38            {
39                velocidad = 0;
40            }
41        }
42
43        velocidad = Mathf.Clamp (velocidad, -1, 3);
44
45        if (velocidad > 0.1f ||velocidad < -0.1f)
46        {
47            this.transform.Rotate(0,Input.GetAxis("Horizontal")*velocidadRotacion*Time.deltaTime,0,Space.Self);
48        }
```

Figura 71. Código para controlar la rotación del modelo

8. Una vez que se ha calculado la *velocidad* dependiendo de la entrada del teclado, se la asigna al Animation Controller con la siguiente línea de código animator.SetFloat(“velocidad”, *velocidad*).

```
16    // Update is called once per frame
17    void Update () {
18        float verticalAxis = Input.GetAxis ("Vertical");
19        if (verticalAxis > 0)
20        {
21            velocidad += aceleracion * Time.deltaTime;
22        }
23        else if (verticalAxis < 0)
24        {
25            velocidad -= aceleracion * Time.deltaTime;
26        }
27        else
28        {
29            if (velocidad > 0.15f)
30            {
31                velocidad -= aceleracion / 2 * Time.deltaTime;
32            }
33            else if(velocidad<-0.15f)
34            {
35                velocidad += aceleracion / 2 * Time.deltaTime;
36            }
37            else
38            {
39                velocidad = 0;
40            }
41        }
42
43        velocidad = Mathf.Clamp (velocidad, -1, 3);
44
45        if (velocidad > 0.1f ||velocidad < -0.1f)
46        {
47            this.transform.Rotate(0,Input.GetAxis("Horizontal")*velocidadRotacion*Time.deltaTime,0,Space.Self);
48        }
49
50        animator.SetFloat ("velocidad", velocidad);
51    }
```

Figura 72. Código para asignar la velocidad calculada al animator

9. Guardar cambios y regresar a Unity. Para probar el funcionamiento del modelo recién programado clic en ► en la Vista la Escena lo que mostrará el personaje, al cual se lo puede controlar con las flechas del teclado.

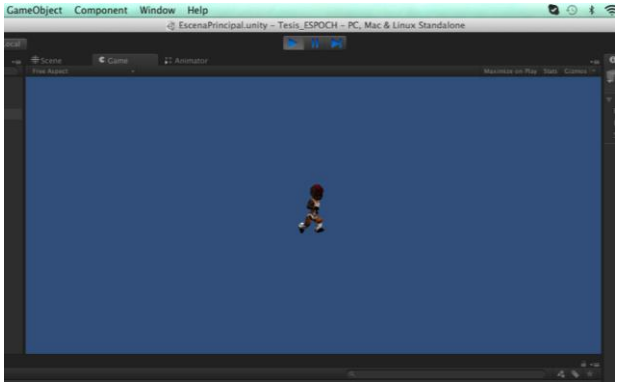


Figura 73. Modelo controlado con el teclado

5.3.11.Añadir a la animación interacción con la física.

1. En la pestaña *Game Object*, dentro de *Create Other* seleccionar *Plane*.

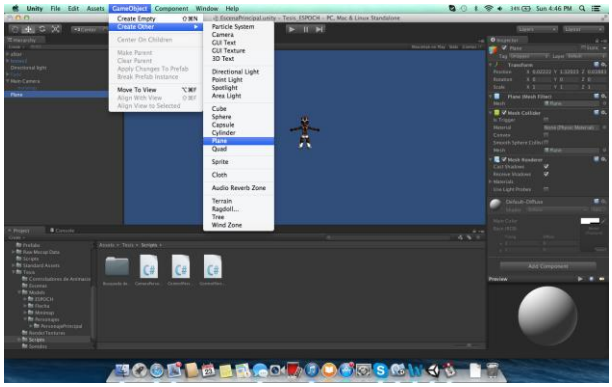


Figura 74. Crear un plano

2. En la Vista de Jerarquía renombrar el *Plane* recién creado a *Piso* dando un clic.

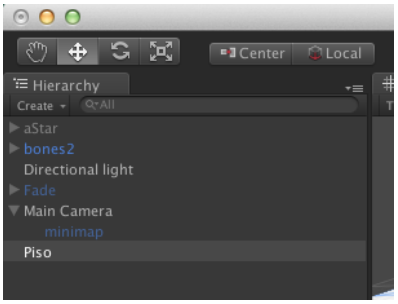


Figura 75. Renombrar el objeto Plano

3. En la *Vista de Escena* configurar el plano cambiando los valores del Piso en posición y escala en el Inspector, para que éste quede debajo del personaje y que sea lo suficientemente extenso para que el personaje pueda moverse sobre él. La escala del *Piso* dependerá de la escala que se le haya dado al personaje.

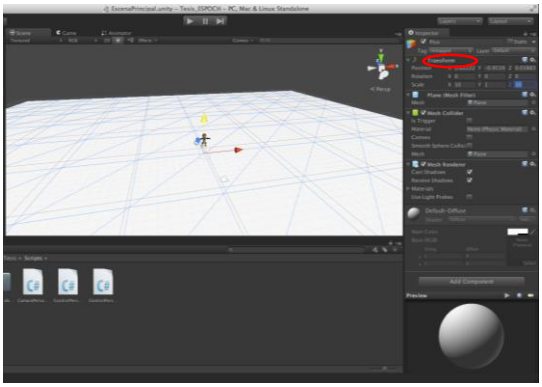


Figura 76. Cambiar los valores del Piso

4. En la pestaña *Game Object* dentro de *Create Other* seleccionamos *Cube* lo que permite crear obstáculos para el movimiento del personaje y poder probar las colisiones. Cambiar los valores de posición, escala y rotación en el *Inspector*.

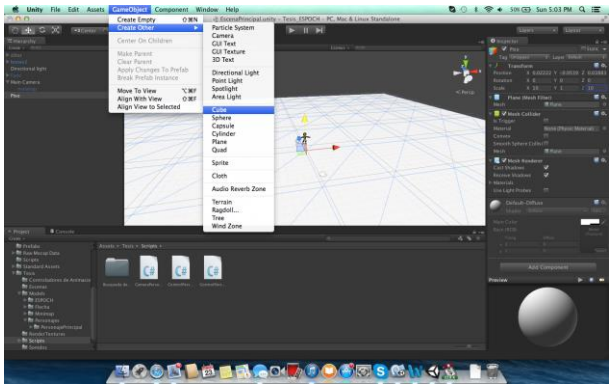


Figura 77. Crear un cubo

5. Renombrar Cube a *Obstáculo* en la *Vista de Jerarquía* y crear más obstáculos de la misma manera que el anterior.

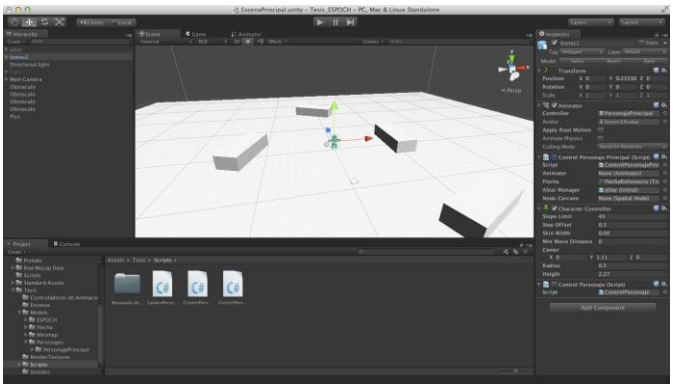


Figura 78. Crear varios obstáculos

6. En la pestaña *Game Object* ir a *Create Other* y seleccionar *Directional* Light, lo que creará una luz direccional que ayudará a apreciar la escena de mejor manera.

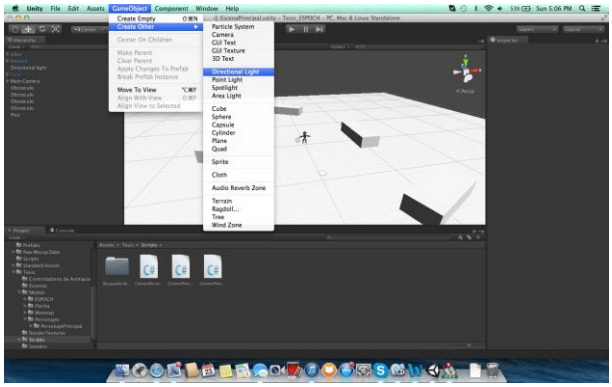


Figura 79. Crear una luz direccional

7. En la Vista de Jerarquía seleccionar al personaje y en la pestaña Componente ir a *Physics* y seleccionar *Character* Controller.

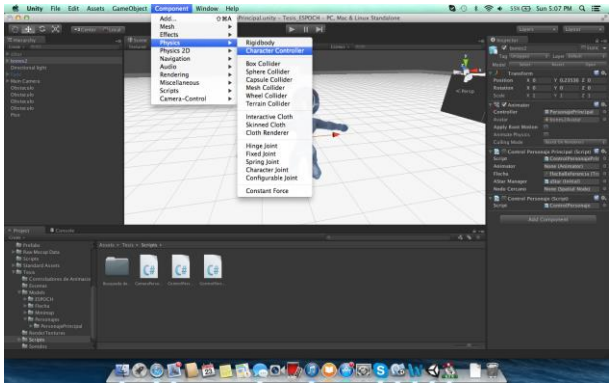


Figura 80. Crear un Character Controller

8. En la parte inferior del *Inspector* se encuentra ubicado el *Character Controller*; configurar los valores de X, Y, Z, radio y altura para que la cápsula que se crea envuelva a todo el personaje. Esta cápsula permitirá el manejo de las colisiones.

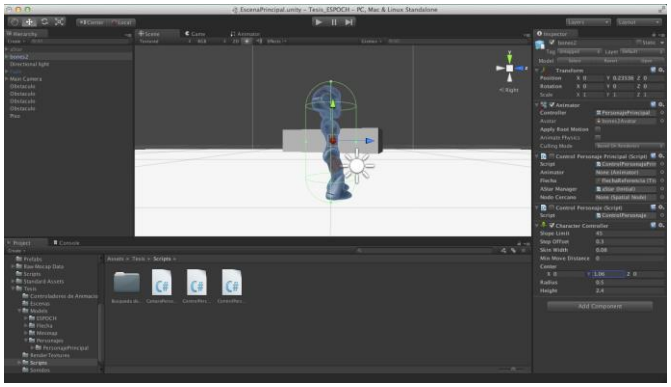


Figura 81. Configurar los valores del Character Controller

9. Volver a MonoDevelop y en el Script de ControlPersonajePrincipal crear una nueva variable de tipo Character Controller con el nombre *controller*.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class ControlPersonaje : MonoBehaviour {
5     private Animator animator;
6     CharacterController controller;
7
8     private float velocidad = 0;
9     private float velocidadRotacion = 30;
10    private float aceleracion = 1.5f;
11    // Use this for initialization
```

Figura 82. Código para crear una variable de tipo Character Controller

10. En el método Start referenciar la variable *controller* con el componente controller por medio de la siguiente línea de código controller = this.GetComponent<CharacterController>().

```
void Start () {
    animator = this.GetComponent<Animator> ();
    controller = this.GetComponent<CharacterController> ();
}
```

Figura 83. Código para referenciar la variable al componente controller

11. En el método Update borrar la línea de código en la que se asigna el valor de velocidad a la animación.

```
if (velocidad > 0.1f ||velocidad < -0.1f)
{
    this.transform.Rotate(0,Input.GetAxis("Horiz
}

animator.SetFloat ("velocidad", velocidad);
```

Figura 84. Borrar el código de asignación del valor de velocidad

12. Para mover el personaje se reemplazará la línea de código anterior con otra en la cual se utiliza la función controller.SimpleMove, la cual recibe un vector que simulará la velocidad; esta función ignorará el movimiento en el eje pero en su lugar tomará en cuenta la gravedad para el movimiento final del modelo. Esta función por lo tanto permitirá controlar la colisión del modelo con los objetos de la escena, impidiendo el movimiento cuando el modelo ha chocado contra algo en cualquiera de los ejes.

```
velocidad = Mathf.Clamp (velocidad, -1, 3);

if (velocidad > 0.1f ||velocidad < -0.1f)
{
    this.transform.Rotate(0,Input.GetAxis("Horizontal")*velocidadRotacion*Time.deltaTime,0,Space.Self);
}

controller.SimpleMove (this.transform.forward * velocidad);
```

Figura 85. Código que permite el movimiento y controla las colisiones

13. Si la variable *velocidad* es positiva (es decir si el modelo se mueve hacia adelante), se asignará el valor de la misma a la animación, utilizando la magnitud del vector velocidad obtenida con controller.velocity.sqrMagnitude; si la variable *velocidad* es negativa (es decir si el modelo se mueve hacia atrás), se le asigna únicamente el valor de la variable *velocidad*.

```
if (velocidad > 0.1f ||velocidad < -0.1f)
{
    this.transform.Rotate(0,Input.GetAxis("Horizontal")*velocidadRotacion*Time.deltaTime,0,Space.Self);
}

controller.SimpleMove (this.transform.forward * velocidad);

if (velocidad > 0)
{
    animator.SetFloat ("velocidad", controller.velocity.sqrMagnitude);
}
else
{
    animator.SetFloat ("velocidad", velocidad);
}
```

Figura 86. Asignar el valor a la variable velocidad

14. Guardar cambios y regresar a Unity. Para probar el funcionamiento de las colisiones clic en ► en la Vista la Escena con lo cual se probará que el modelo colisiona con los objetos.

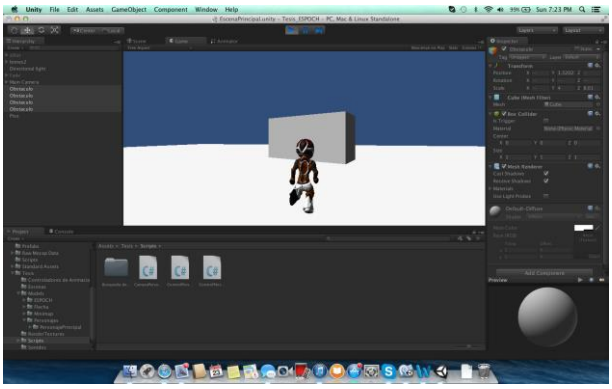


Figura 87. Prueba del funcionamiento de las colisiones

5.3.12.Creación del mini mapa

1. En la pestaña *Game Object*, dentro de *Create Other* seleccionar *Camera*.

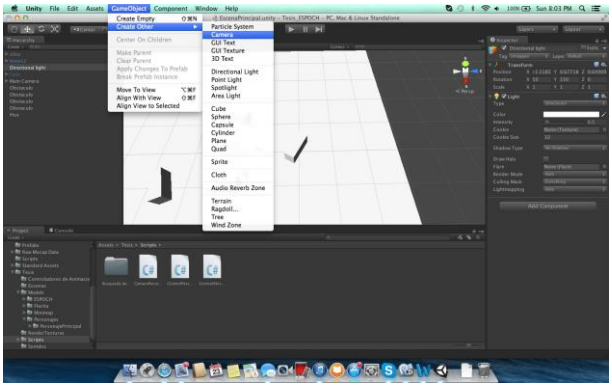


Figura 88. Crear una cámara

2. En la Vista de Jerarquía cambiar el nombre a *MiniMapaCamara*.

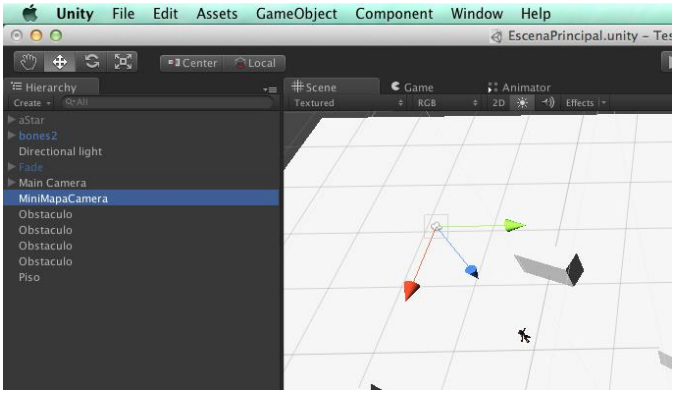


Figura 89. Renombrar a MiniMapaCamara

3. En la Vista de Proyecto dentro de “Assets” en la carpeta “Tesis” crear carpeta que se llame “Render Textures”. Clic Derecho seleccionar *Create* y elegimos *Render Texture*.

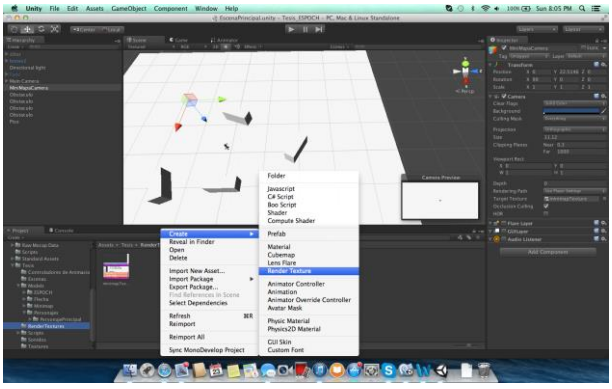


Figura 90. Crear un Render Texture

4. Renombramos a Render Texture como *minimapTextura*.

Figura 91. Renombrar la Textura a minimapTextura

5. Marcar en la Vista de Jerarquía *MiniMapaCamara* y en el *Inspector* buscar *Target Texture* y seleccionar la textura creada recientemente.

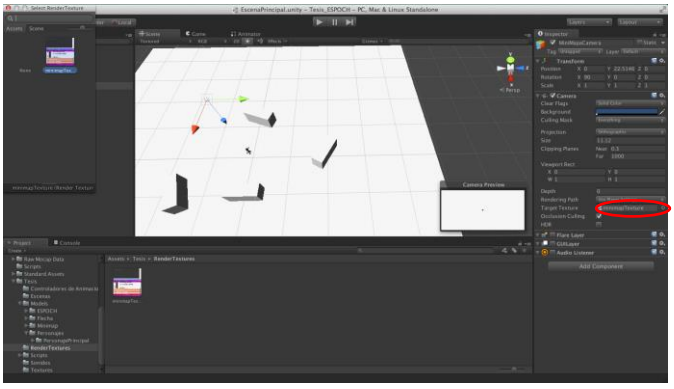


Figura 92. Agregar la textura creada en Target Texture

6. Ubicar la cámara en la misma posición del modelo en el eje X y en el eje Z, y en el eje Y ubicarla sobre el objeto. Para esto revisar los valores que se le asignaron anteriormente al modelo en el Inspector. El valor de la rotación en el eje X es de 90 y en los ejes Y y Z es de 0.

Figura 93. Ubicar la cámara sobre el modelo

7. En el campo *Clear Flags* del Inspector seleccionar Solid Color.

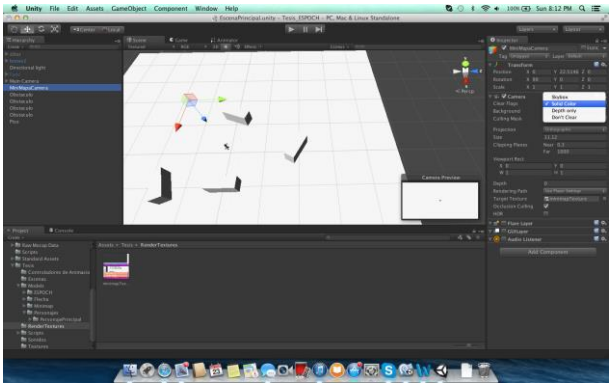


Figura 94. En la opción Clear Flags seleccionar Solid Color

8. Asignar Orthographic a la propiedad *Projection* igualmente en el *Inspector*.

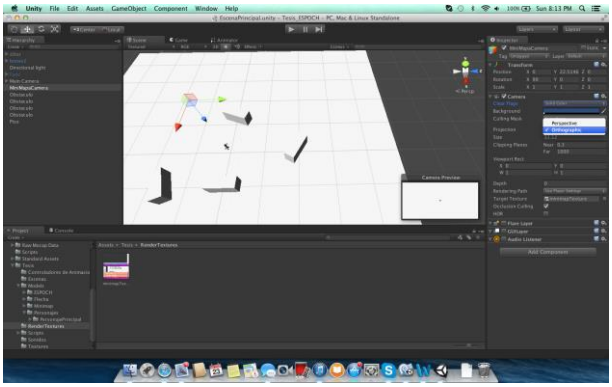


Figura 95. En la opción Projection seleccionar Orthographic

9. Por último ajustar el tamaño de la cámara cambiando el valor de *Size* a un valor que sea conveniente, por ejemplo 9.

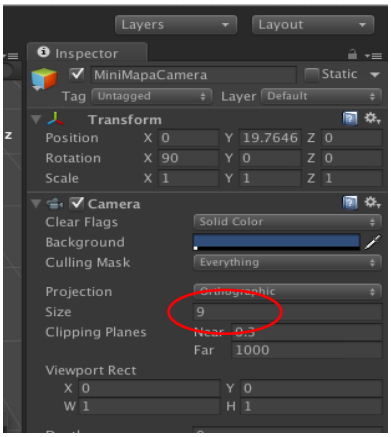


Figura 96.Ajustar el tamaño de la cámara

10. Volver a la *Vista de Jerarquía* y arrastrar la cámara *MiniMapCamara* dentro del modelo, para que la cámara se mueva siempre sobre el personaje.

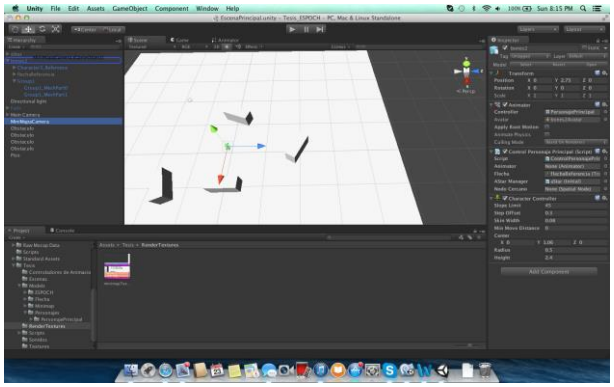


Figura 97. Integrar la cámara con el modelo

11. En la *Vista de Proyecto* dentro de la carpeta “Tesis” ubicada en “Assets” crear una nueva carpeta llamada “Materiales”. Clic derecho seleccionar *Create* y elegir *Material*.

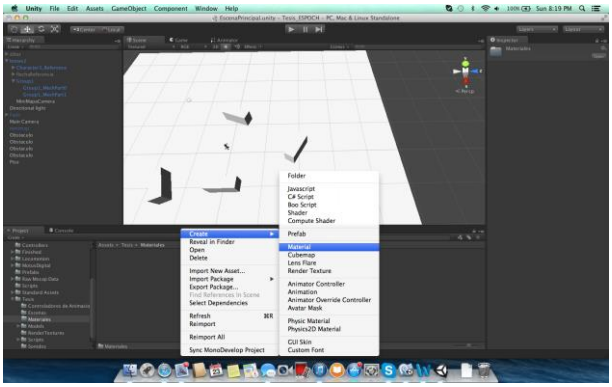


Figura 98. Crear un nuevo material

12. Renombrar el material a *ColorCubos*.

Figura 99. Renombrar el material a ColorCubos

13. En el *Inspector* en *Main Color* escoger un color para asignarles a los Obstaculos.

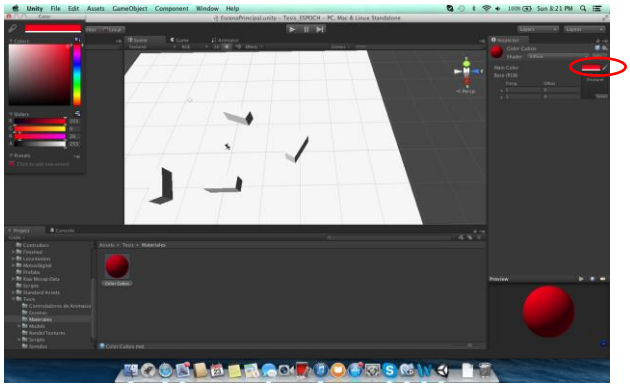


Figura 100. Asignar color a los obstáculos

14. En la Vista de Jerarquía seleccionar todos los *Obstaculos* y arrastrar el color desde la carpeta “Materiales” hacia en botón *Add Component* en la parte inferior del *Inspector*.

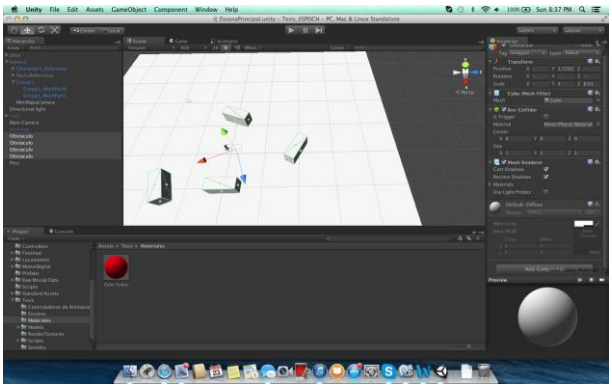


Figura 101. Asignar el material a todos los obstáculos

15. En la pestaña *Game Object* y dentro de *Create Other* seleccionar *GUI Texture.*

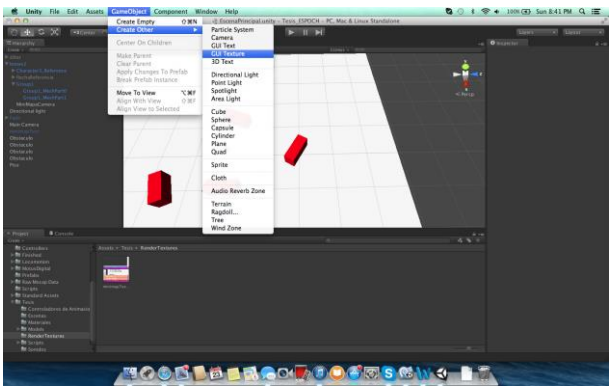


Figura 102. Crear un GUI Texture

16. Renombrar el objeto creado a *MiniMap* y en el Inspector en la opción *Texture* seleccionar el *minimapTextura* que se creó anteriormente.

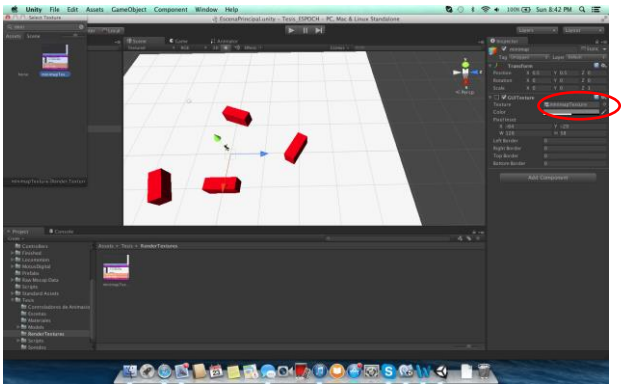


Figura 103. Asignar el minimapTexture

17. En el Inspector en Color cambiar el valor de *A*(alpha) al más alto que es 255.

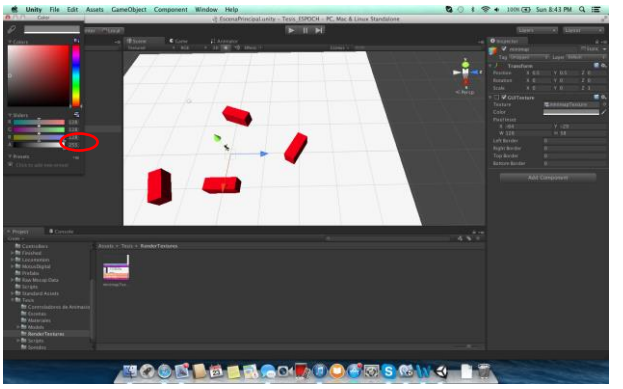


Figura 104. Cambiar el valor de Alpha

18. Cambiar el tamaño del *miniMap* en el Inspector cambiando los valores de X a -100 Y a -75 W a 200 y H a 150 del *Pixel Inset*; y cambiar los valores de la posición del *miniMap* en *Transform* asignando un valor de 0.87 a X y 0.8 a Y.

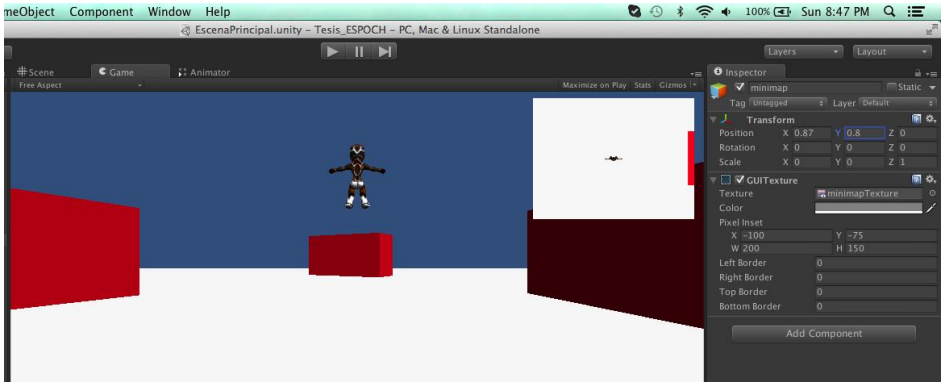


Figura 105. Cambiar la configuración del miniMap

19. Para probar el funcionamiento correcto del mini mapa clic en ► en la Vista la Escena con lo cual se comprobará que el mini mapa funciona correctamente y facilita la ubicación del personaje en la escena.

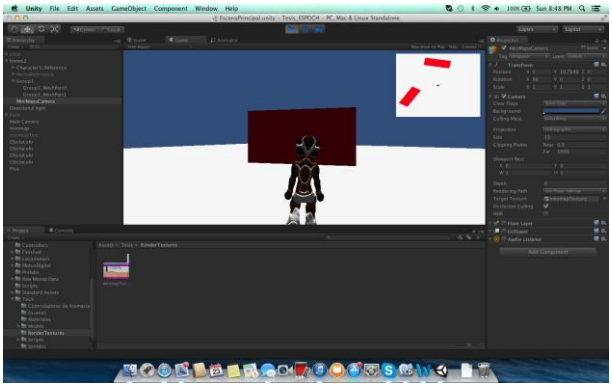


Figura 106. Mini mapa en funcionamiento

5.3.13.Implementación de la búsqueda de caminos

Para implementar la búsqueda de caminos se necesita de un algoritmo de búsqueda, en este caso se utilizará el algoritmo de búsqueda A*; ya que bajo ciertas condiciones, encuentra el camino más corto entre un nodo de origen y uno de destino.

Para esta implementación se seguirán los siguientes pasos:

1. Crear una nueva carpeta en la Vista de Proyecto dentro de la carpeta existente “Scripts” a la que se la llamará “Búsqueda de Caminos”.
   Dentro de ella dar clic derecho y crear un nuevo Script al que se le llamará *NodoEspacial*.

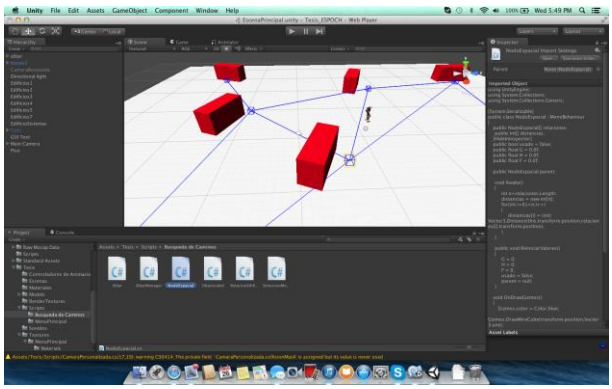


Figura 107. Crear el Script NodoEspacial

2. Abrir el Script en Mono Develop y declarar las siguientes variables públicas y de tipo flotante: *G, H* y *F* inicializadas en 0. Declarar además un vector de nodoEspacial llamado *relaciones* utilizado para almacenar con que otros nodos se relaciona un nodo actual y ya que se buscará el camino más corto por distancia se utilizará una variable de tipo vector de enteros llamado *distancias* para almacenar la distancia del nodo actual con los que se relacione. Se necesitará una última variable de tipo NodoEspacial llamada *parent* para almacenar el nodo anterior al actual en el camino una vez que se ha encontrado el más óptimo

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

[System.Serializable]
public class NodoEspacial : MonoBehaviour
{
    public NodoEspacial[] relaciones;
    public int[] distancias;

    public float G = 0.0f;
    public float H = 0.0f;
    public float F = 0.0f;

    public NodoEspacial parent;

```

Figura 108. Código para crear variables en NodoEspacial

3. Crear el método privado Awake el cual se ejecuta antes del método Start, dentro de él se llenará en vector *distancias* haciendo el cálculo entre la posición del nodo actual con cada uno de los nodos con los que ésta relacionado; para esto se utilizará una estructura de repetición que recorrerá todo el vector de relaciones y realizará el cálculo utilizando la función Vector3.Distance.

```
void Awake()
{
    int n=relaciones.Length;
    distancias = new int[n];
    for(int i=0;i<n;i++)
    {
        distancias[i] = (int) Vector3.Distance(this.transform.position,relaciones[i].transform.position);
    }
}
```

Figura 109. Código para crear el método Awake

4. Se crea el método ReiniciarValores el cual asigna los valores iniciales a *F, G, H* y *parent* para iniciar una nueva búsqueda ya que estas variables tomarán nuevos valores cada que se realice una búsqueda.

```
public void ReiniciarValores()
{
    G = 0;
    H = 0;
    F = 0;

    parent = null;
}
```

Figura 110. Código para crear el método ReiniciarValores

5. Por último se crea el método OnDrawGizmos el cual es propio de Unity y sirve para la depuración. Se usará este método para dibujar líneas entre el nodo y todos los nodos con los que se relacione, estas líneas serán vistas en la Vista de Escena mas no en la Vista de Juego.

   Se asigna un color para dibujar las líneas con el código Gizmos.color = Color.blue. Se dibuja un cubo en la posición del nodo con el código Gizmos.DrawWireCube(transform.position,Vector3.one); si el vector *relaciones* es diferente de null entonces se recorre cada una de sus posiciones dibujando una línea hacia ellas con el código Gizmos.DrawLine(transform.position, relaciones[i].transform.position).

```
void OnDrawGizmos()
{
    Gizmos.color = Color.blue;
    Gizmos.DrawWireCube(transform.position,Vector3.one);
    if (relaciones != null)
    {
        for (int i = 0; i < relaciones.Length; i++)
        {
            if (relaciones[i] != null)
            {
                Gizmos.DrawLine(transform.position, relaciones[i].transform.position);
            }
        }
    }
}
```

Figura 111. Código para crear el método OnDrawGizmos

6. Crear un *Game Object* vacío. Para esto en la pestaña *Game Object* seleccionar *Create Empty*.

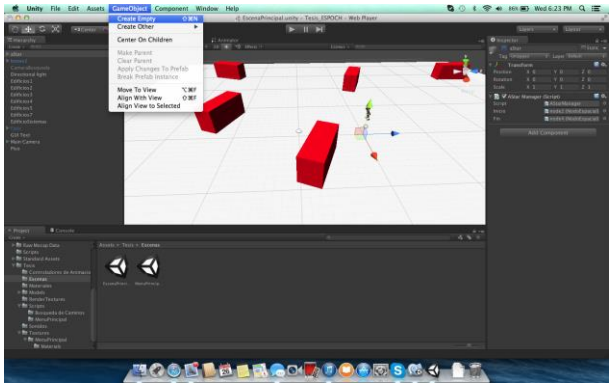


Figura 112. Crear un Game Objet vacío

7. Renombrarlo como *nodo1* en la Vista de Jerarquía y añadirle el Script nodoEspacial en el *Inspector* como se ha visto anteriormente

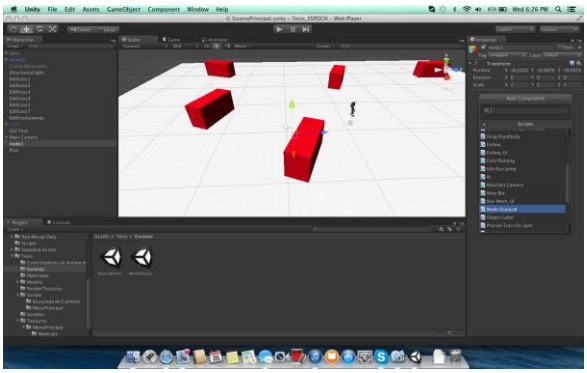


Figura 113. Agregar el Script nodoEspacial al Game Object

8. Duplicar el nodo1 varias veces, cada nodo es un punto en la búsqueda. En cada nodo ir al Inspector, en el campo *Relaciones* opción *Size* asignar el número de nodos con los que ésta relacionado; en cada uno de las opciones *Element* asignar los nodos con los que se relaciona. En la Vista de Escena se verá como se dibujan las líneas de relación de los nodos.

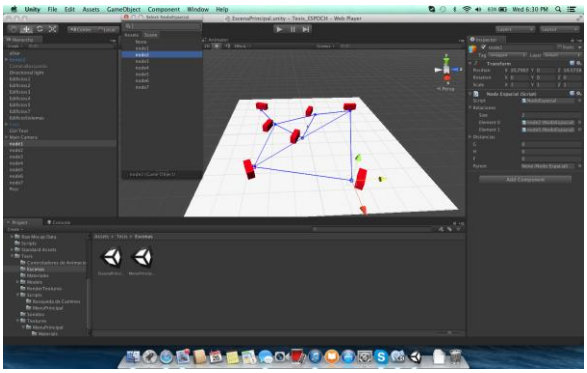


Figura 114. Crear varios nodos

9. Volver a la Vista de Proyecto y crear en la carpeta “Busqueda de Caminos” un Script llamado AStar.

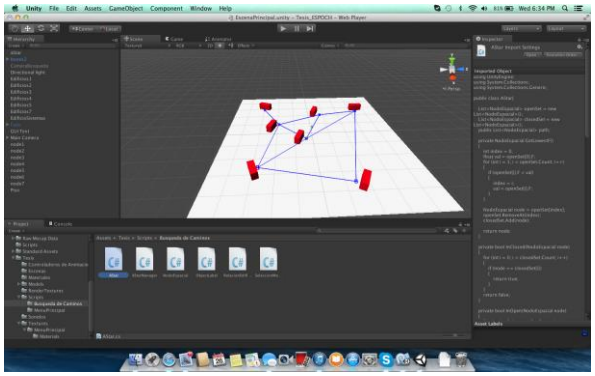


Figura 115. Crear el Script AStar

10. Abrir el Script creado en Mono Develop e implementar el algoritmo AStar (Anexo *2)*, que es la implementación de PseudoCódigo que se encuentra en el punto 3.12.1 del presente documento. Este algoritmo se encargará de encontrar el camino más corto para llegar de un nodo a otro, pero está optimizado para tratar de encontrar el camino buscando siempre que el nodo siguiente se encuentre más cerca del nodo buscado; por lo tanto siempre elegirá como nodo siguiente el que se encuentre enlazado con la distancia más corta al nodo final. Este Script interactuará con otros para obtener los valores necesarios para devolver el camino final.
11. Guardar cambios y regresar a Unity para crear un nuevo Script llamado AStarManager. Doble clic en él y en Mono Develop declarar las variables públicas de tipo NodoEspacial *inicio* y *fin;* un vector privado de tipo NodoEspacial llamado *nodos* y una de tipo AStar llamada *aStar* a la cual se le inicializa. Se crea además una variable de tipo entero llamada *indiceActual* inicializada en 0.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class AStarManager : MonoBehaviour {
5
6     public NodoEspacial inicio;
7     public NodoEspacial fin;
8
9     private NodoEspacial[] nodos;
10    AStar aStar = new AStar();
11
12    int indiceActual = 0;
13
```

Figura 116. Código para crear variables en AStarManager

12. En el método Start se buscarán todos los hijos que tenga este componente, todos ellos deberán tener añadido el componente nodoEspacial. Cada uno de los hijos deberá estar añadido en el vector *nodos* para tener una referencia rápida a estos.

```
void Start()
{
    int numOfChilds = this.transform.childCount;
    nodos = new NodoEspacial[numOfChilds];

    for(int i=0;i<numOfChilds;i++)
    {
        nodos[i] = this.transform.GetChild(i).GetComponent<NodoEspacial>();
    }
}
```

Figura 117. Código para buscar los hijos de un nodo

13. Crear el método ReiniciarValoresNodos el cual utilizará una estructura de repetición foreach para reiniciar cada uno de los nodos almacenado en el vector *nodos* a su valor inicial.

```
void ReiniciarValoresNodos()
{
    foreach (NodoEspacial node in nodos)
    {
        node.ReiniciarValores();
    }
}
```

Figura 118. Código para ReiniciarValoresNodos

14. El siguiente método a crear es BuscarCamino el que recibe como parámetro de entrada un NodoEspacial *inicio.* En este método se reinician todos los nodos a sus valores iniciales y se ejecuta el algoritmo de búsqueda (Script AStar) para luego asignarse a *indiceActual* el valor del último índice que tiene el camino devuelto por el algoritmo; esto se debe a que el camino que devuelve empieza desde el final hasta el inicio.

```
public void BuscarCamino(NodoEspacial inicio)
{
    this.inicio = inicio;
    ReiniciarValoresNodos ();
    aStar.Start (this.inicio, fin);
    indiceActual = aStar.path.Count;
}
```

Figura 119. Crear el método BuscarCamino

15. Se crea, como siguiente paso, la función GetSiguienteNodoEnCamino de tipo NodoEspacial. Esta función devolverá el siguiente nodo al actual hasta llegar al último nodo cuando devolverá el valor de null.

```
public NodoEspacial GetSiguienteNodoEnCamino()
{
    indiceActual--;
    if (indiceActual>=0)
    {
        return aStar.path[indiceActual];
    }
    return null;
}
```

Figura 120. Crear la función GetSiguienteNodoEnCamino

16. Crear una siguiente función llamada GetIndiceCercano, la que recibe como parámetro una *posicion* de tipo Vector3 y retorna el índice del nodo que se encuentre más cercano a esa posición. Se utiliza un algoritmo que busca la distancia menor entre la posición y cada uno de los nodos.

```
public int GetIndiceCercano(Vector3 posicion)
{
    float minDist = float.MaxValue;
    int indice = 0;
    int n = nodos.Length;
    for (int i=0; i<n; i++)
    {
        float distancia = Vector3.SqrMagnitude(posicion-nodos[i].transform.position);
        if(distancia<minDist)
        {
            indice = i;
            minDist = distancia;
        }
    }
    return indice;
}
```

Figura 121. Código para crear la función GetIndiceCercano

17. Se crea una nueva función a la que se nombra GetNodoCercano, la que como parámetro recibe una *posicion* de tipo Vector3 y al igual que en la función anterior realiza el cálculo para buscar la distancia mínima y retorna el nodo en lugar de solamente el índice.

```
public NodoEspacial getNodoCercano(Vector3 posicion)
{
    float minDist = float.MaxValue;
    int indice = 0;
    int n = nodos.Length;
    for (int i=0; i<n; i++)
    {
        float distancia = Vector3.SqrMagnitude(posicion-nodos[i].transform.position);
        if(distancia<minDist)
        {
            indice = i;
            minDist = distancia;
        }
    }
    return nodos[indice];
}
```

Figura 122. Código para crear la función GetNodoCercano

18. Se crea por último OnDrawGizmos que permitirá graficar la ruta encontrada. Recorr el camino dibujando líneas como se vio anteriormente.

```
void OnDrawGizmos()
{
    if (inicio != null)
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireCube(inicio.transform.position, Vector3.one * 1.3f);
    }

    if (fin != null)
    {
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireCube(fin.transform.position, Vector3.one * 1.3f);
    }

    if (aStar.path != null)
    {
        for (int i = 0; i < aStar.path.Count-1; i++)
        {
            Gizmos.color = Color.cyan;
            Gizmos.DrawWireCube(aStar.path[i].transform.position, Vector3.one * 1.5f);
            Gizmos.DrawWireCube(aStar.path[i + 1].transform.position, Vector3.one * 1.5f);
            Gizmos.DrawLine(aStar.path[i].transform.position + new Vector3(0, 1, 0), aStar.path[i + 1].transform.position + new Vector3(0, 1, 0));
        }
    }
}
```

Figura 123. Crear el método OnDrawGizmos

19. Guardar cambios y volver a Unity. Ir a la pestaña *Game Object* y seleccionar *Create Empty*, renombrarlo a aStar y buscar en la opción *Add* Component del *Inspector* el Script AStar Manager

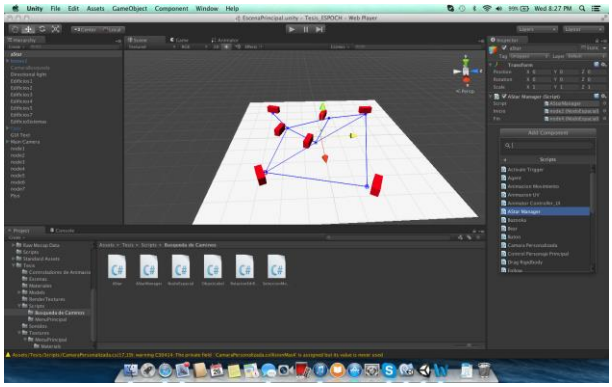


Figura 124. Crear un Game Object y asignarle el Script

20. Seleccionar todos los nodos creados en la *Vista de Jerarquía* y arrastrarlos dentro del objeto aStar para que se conviertan en hijos del objeto y será funcional el Script anterior.

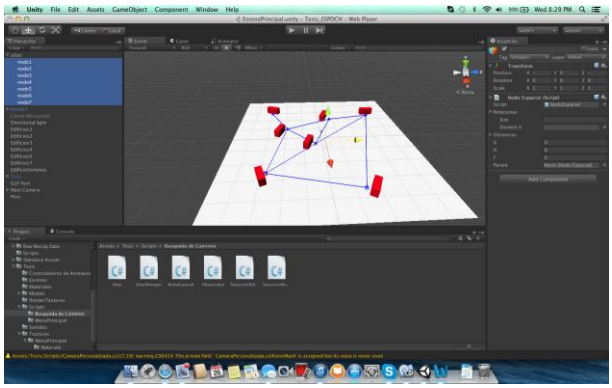


Figura 125. Convertir todos los nodos en hijos de AStar

21. Para probar el funcionamiento, regresar al Script AStarManager y al final del método Start añadimos el código aStar.Start(inicio,fin) lo cual calculará el camino entre un nodo inicio y fin que se le asigne al arrancar la aplicación.

```
void Start()
{
    int numOfChilds = this.transform.childCount;
    nodos = new NodoEspacial[numOfChilds];

    for(int i=0;i<numOfChilds;i++)
    {
        nodos[i] = this.transform.GetChild(i).GetComponent<NodoEspacial>();
    }

    aStar.Start (inicio, fin);
}
```

Figura 126. Código para encontrar el camino entre 2 nodos

22. Regresar a Unity y en la *Vista de Jerarquía* seleccionar el objeto aStar. En el *Inspector* dentro del componente *AStarManager* asignar un nodo inicio y un nodo fin para probar.

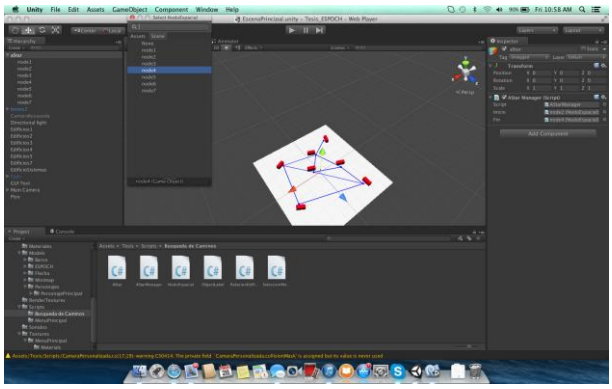


Figura 127. Asignar un nodo inicio y fin en la escena

23. Clic en ► y en la Vista de Escena se dibujará el camino encontrado entre los dos nodos.

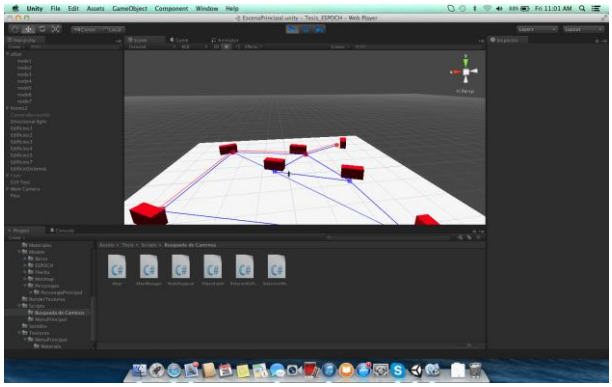


Figura 128. Camino dibujado entre 2 nodos

5.3.14.Interacción del personaje con la búsqueda de caminos

1. Volver a Mono Develop e ingresar al Script ControlPersonajePrincipal. Declarar una variable pública de tipo Transform llamada *flecha* que servirá para indicar el camino correcto hacia el nodo fin. Declarar también una variable pública de tipo AStarManager llamada *aStarManager* que permitirá que se realice la búsqueda. Por último se crea una variable privada de tipo NodoEspacial llamada *nodoCercano* que referenciará al nodo más cercano del camino correcto.

```
public class ControlPersonajePrincipal : MonoBehaviour {
    public Animator animator;
    public Transform flecha;
    public AStarManager aStarManager;

    public NodoEspacial nodoCercano;
    CharacterController controller;

    private float velocidad = 0;
    private float velocidadRotacion = 30;
    private float aceleracion = 1.5f;
```

Figura 129. Código para declarar variables para la interacción de personajes y búsqueda de caminos

2. Se añadirá al principio del método Update el código que buscará el nodo más cercano a la posición actual del personaje cuando se presione la tecla espacio y se asignará a la variable *aStarManager* este nodo cercano como nodo de inicio para la búsqueda de caminos. Se activará también la flecha que apuntará el camino correcto.

```
    // Update is called once per frame
    void Update () {

        if (Input.GetKeyDown (KeyCode.Space))
        {
            nodoCercano = aStarManager.getNodoCercano(this.transform.position);
            aStarManager.BuscarCamino(nodoCercano);
            nodoCercano = aStarManager.GetSiguienteNodoEnCamino();
            flecha.gameObject.SetActive(true);
        }
```

Figura 130. Código para asignar el nodo más cercano a la posición actual como nodo inicio

3. A continuación en el mismo método, se configura la flecha creada para siempre apunte correctamente hacia el nodo siguiente hasta llegar al fin del camino. Se pregunta si el *nodoCercano* es diferente a null, si es así se configurará la flecha para que apunte hacia la posición del *nodoCercano* y se calculará la distancia entre el personaje y el *nodoCercano*; si esta distancia es menor que un valor asignado (en este caso 3), al *nodoCercano* se le asigna el siguiente nodo en el camino.
   Si el *nodoCercano* es igual a null entonces se desactiva la flecha.

```
if (nodoCercano != null)
{
    flecha.LookAt(nodoCercano.transform);
    if(Vector3.Distance(this.transform.position,nodoCercano.transform.position)<3)
    {
        nodoCercano = aStarManager.GetSiguienteNodoEnCamino();
    }
}
else
{
    flecha.gameObject.SetActive(false);
}
```

Figura 131. Código para configurar la flecha que dirige el personaje al nodo final

4. Guardar cambios y regresar a Unity. Importar el modelo de una flecha como se ha visto anteriormente y llevarlo hacia la *Vista de Jerarquía*. Renombrarla a *flechaReferencia*. Ir al Inspector y configurar los valores para que se ubique sobre el personaje.

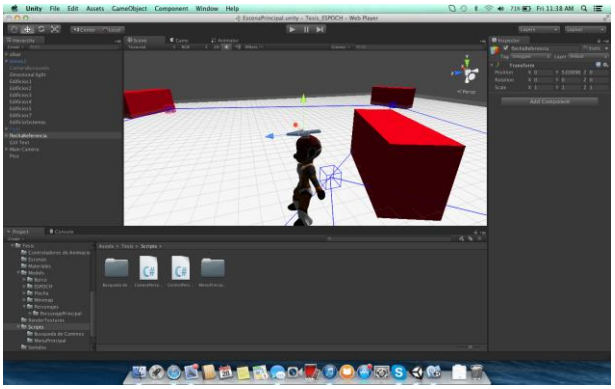


Figura 132. Configurar ubicación de la flecha sobre el personaje

5. En la Vista de Jerarquía arrastrar el objeto *flechaReferencia* dentro del personaje principal para que ésta siempre se ubique sobre el personaje.

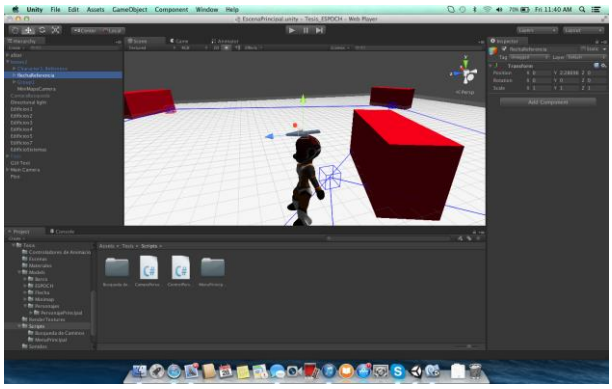


Figura 133. Arrastrar la flecha dentro del personaje

6. En la *Vista de Jerarquía* se selecciona el personaje y en el *Inspector* se busca el componente *ControlPersonajePrincipal*. En el campo Flecha seleccionar *flechaReferencia* y en el campo *AStarManager* se selecciona aStar.

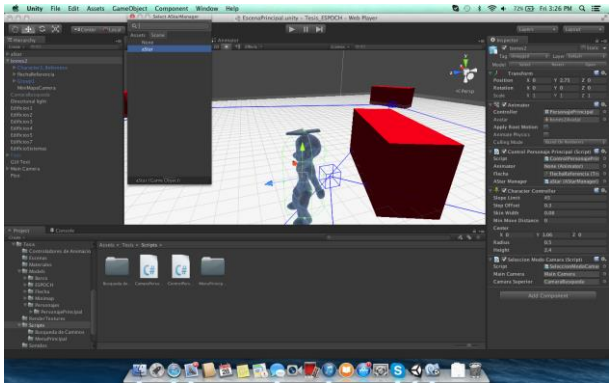


Figura 134. Asignar los componentes de control de personaje

7. Seleccionar en la *Vista de Jerarquía* el objeto *aStar* y en el *Inspector* en el componente *AStar Manager* buscar el campo Fin y seleccionar cualquiera de los nodos previamente creados.

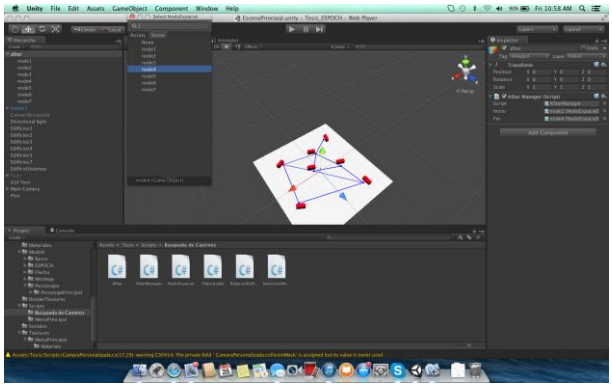


Figura 135. Asignar un nodo fin para probar el funcionamiento

8. Clic en ► para probar el funcionamiento. Al presionar la tecla *espacio* la flecha aparecerá sobre el personaje y señalará el camino correcto hacia el nodo asignado previamente como fin.

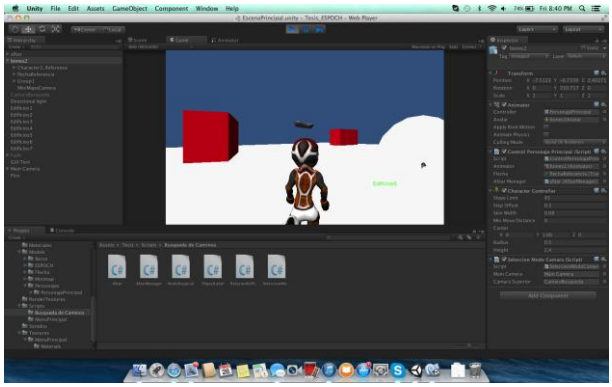


Figura 136. Flecha sobre el personaje indiacndo el camino hacia el nodo final

9. En la pestaña *Game Object* seleccionar *Create Other* y elegir *Camera*. Renombrar el objeto creado en la *Vista de Jerarquía* a *CamaraBusqueda*.

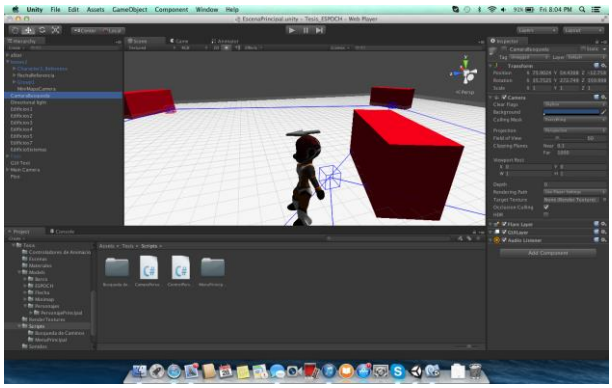


Figura 137. Crear una nueva cámara

10. En la Vista de Escena con el mouse mover toda la escena creada hasta que se visualice por completo. En la Vista de Jerarquía seleccionar *CamaraBusqueda*, ir a la Pestaña *Game Object* y clic en *Align With View*. De esta forma la cámara se posiciona y rota para que coincida con la que se ve en la *Vista de Escena*.

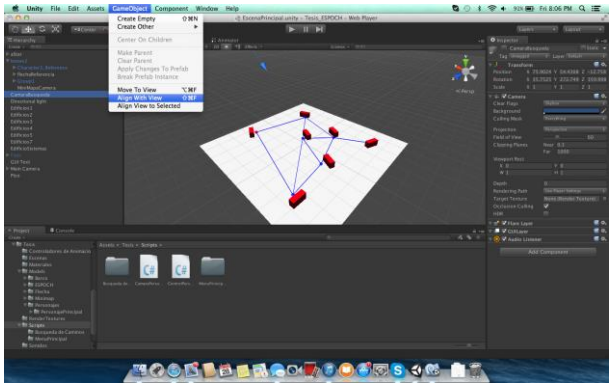


Figura 138. Posicionar la cámara Superior sobre la escena

11. En la *Vista de Jerarquía* seleccionar *CamaraBusqueda* y en el *Inspector* deseleccionar el componente *CamaraBusqueda* que se encuentra en la parte superior.

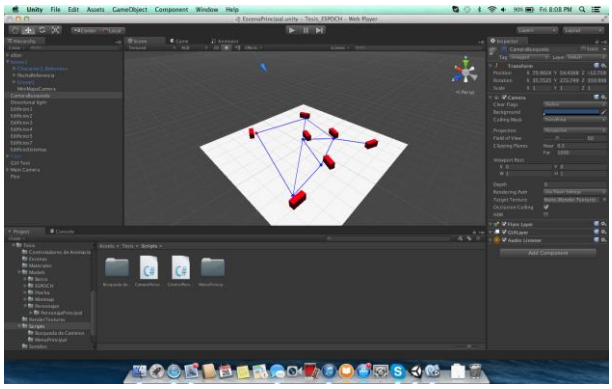


Figura 139. Desactivar la cámara Superior creada

12. Crear un nuevo Script llamado *SeleccionModoCamara* en el cual se permitirá al usuario seleccionar la cámara con la que desea ver el Mundo Virtual. Para esto se declaran dos variables públicas de tipo *GameObject*, la primera llamada *mainCamera* y la otra *camaraSuperior*.

```
public class SeleccionModoCamara : MonoBehaviour {
    public GameObject mainCamera;
    public GameObject camaraSuperior;
```

Figura 140. Código para crear variables para cada cámara

13. En el método Update se controlará que al presionar en el teclado la letra “C” el estado activo de la cámara actual se desactive activando la otra cámara existente.

```
// Update is called once per frame
void Update () {
    if(Input.GetKeyDown(KeyCode.C))
    {
        mainCamera.SetActive(!mainCamera.activeSelf);
        camaraSuperior.SetActive(!camaraSuperior.activeSelf);
    }
}
```

Figura 141. Código para controlar la cámara deseada

14. Guardar cambios y volver a Unity. Crear un nuevo Script llamado RelacionEdificioNodo, el que servirá para determinar que nodo es el más cercano a cada edificio. Se crearán dos variables públicas, la primera de tipo NodoEspacial llamada *nodoCercano* y la otra es de tipo AStarManager y se la llamará aStarManager.

```
public class RelacionEdificioNodo : MonoBehaviour {
    public NodoEspacial nodoCercano;
    public AStarManager aStarManager;
```

Figura 142. Código para crear variables para asignar un nodo a un edificio

15. Añadir el evento OnMouseDown el que permitirá que cuando se dé clic sobre el edificio al que se le agregará este Script, se designe como Fin para la búsqueda de camino el nodo más cercano al edificio.

```
void OnMouseDown()
{
    aStarManager.fin = nodoCercano;
}
```

Figura 143. Código para permitir que cuando se de clic en un edificio se añada como nodo fin la búsqueda

16. Volver a Unity y en la Vista de Jerarquía renombrar todos los objetos llamados obstáculo por Edificio y numerarlos.

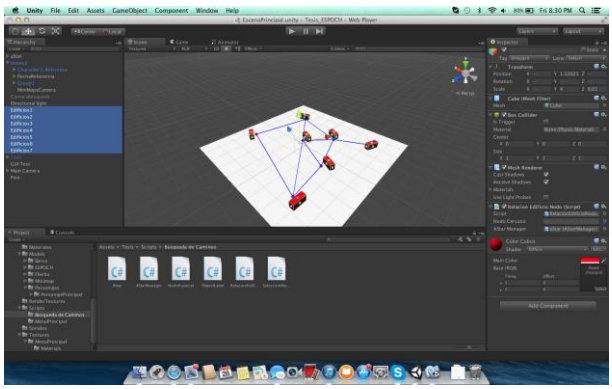


Figura 144. Renombrar los obstáculos a edificio#

17. Con todos los edificios seleccionados, en el Inspector en *Add Componet* seleccionar Scripts y buscar *RelacionEdificioNodo*.

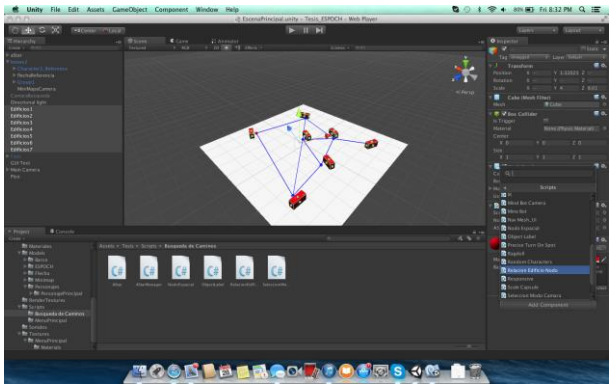


Figura 145. Añadir a los edificios el Script de RelacionNodoEdificio

18. Seleccionar en la Vista de Jerarquía cada uno de los edificios y en el Inspector buscar el componente Relacion Edificio Nodo. En el campo *AStar Manager* asignar el objeto *aStar* y en el campo *Nodo Cercano* asignar el nodo que se encuentre más cercano al edificio.

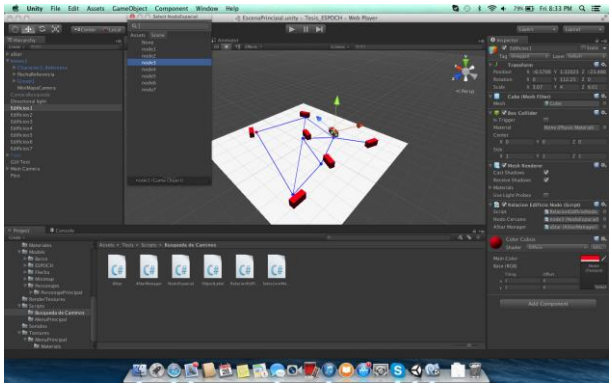


Figura 146. Añadir a los edificios el nodo más cercano

19. En la *Vista de Jerarquía* seleccionar el personaje y añadir el Script *SeleccionModoCamara* en *Add Component* en el *Inspector*.

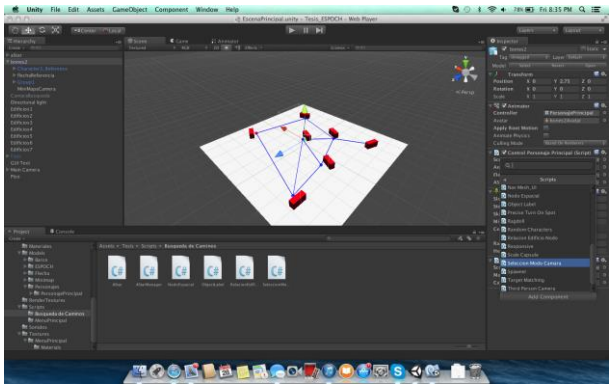


Figura 147. Añadir al personaje el Script para el manejo de cámaras

20. En el componente Selección Modo Camera; en la campo *Main Camara* asignar el objeto Main Camera y en el campo *Camara Superior* asignar el objeto *Camara Busqueda*.

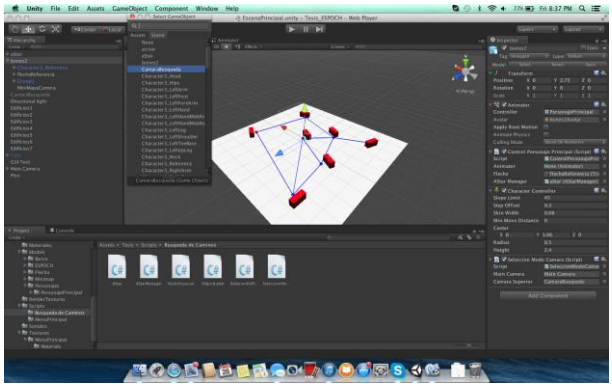


Figura 148. Asignar cada cámara a su campo correspondiente

21. Clic en ► y presionar “C”. Se puede apreciar el cambio de cámara de la Principal a la Superior y viceversa. Si se le da clic a un edificio y se presiona la tecla espacio, la flecha aparecerá y guiará al personaje al edificio seleccionado.

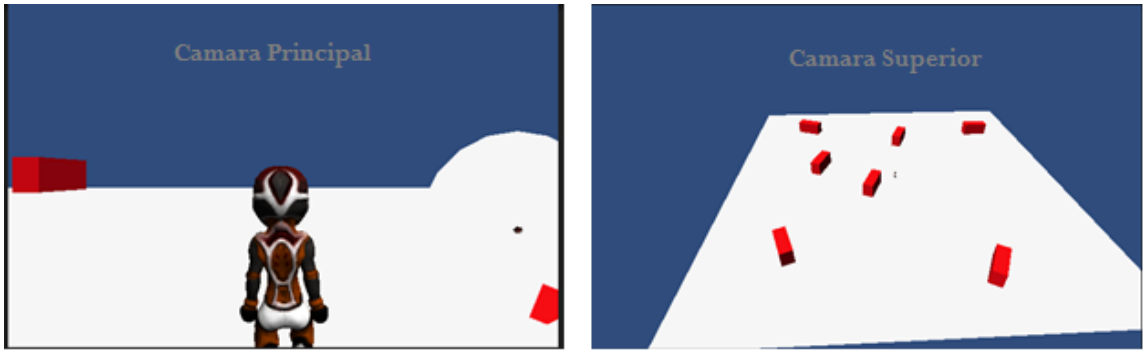


Figura 149. Vista de cada cámara

5.3.15.Configuración de la cámara Principal

Unity por defecto contiene varios Scripts y componentes que son útiles y pueden ser usados en la mayoría de proyectos. Uno de estos es Smooth Follow que permite que la cámara principal siga al personaje suavemente tanto en posición como en rotación.

1. En la pestaña Assets ir a *Import Package* y escoger *Scripts*.

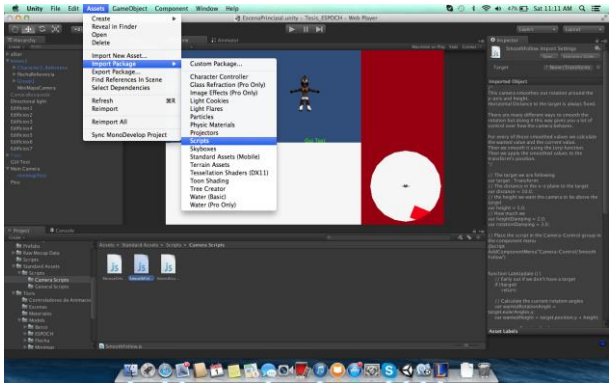


Figura 150. Importar un Script de Unity

2. En la ventana llamada *Importing Package* seleccionar todas las opciones que se muestran y dar clic en *Import*.

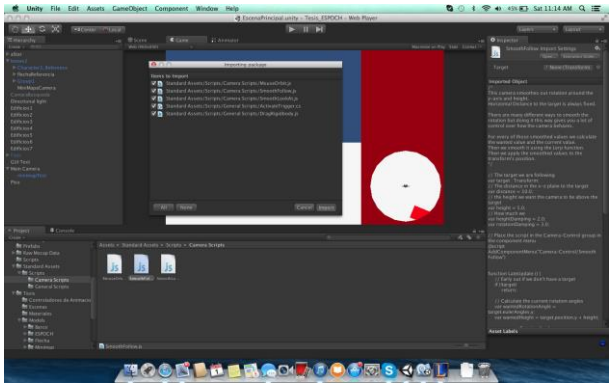


Figura 151. Seleccionar todos los paquetes a importar

3. En la Vista de Proyecto buscar en la carpeta “Assets” una subcarpeta llamada “Standard Assets”, en la cual se encuentra una carpeta llamada “Scripts” donde se ubica “Camera Scripts” dentro de la cual se ubican los Scripts que se pueden usar para configurar la cámara.

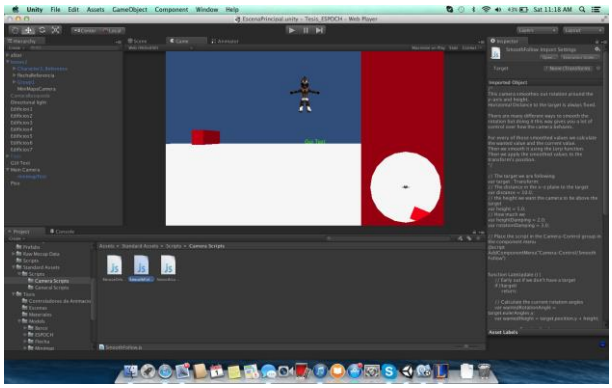


Figura 152. Ubicar un Script de cámara

4. Buscar en la Vista de Jerarquía el objeto *Main Camera*, ir al *Inspector* y en *Add Component* en la opción *Camera-Control* se escoge *Smooth Follow*.

Figura 153. Añadir el Script a la cámara

5. En la Vista de Jerarquía seleccionar nuevamente *Main Camera* y buscar el componente *Smooth Follow* en el Inspector, en el campo *Target* se selecciona el personaje principal.

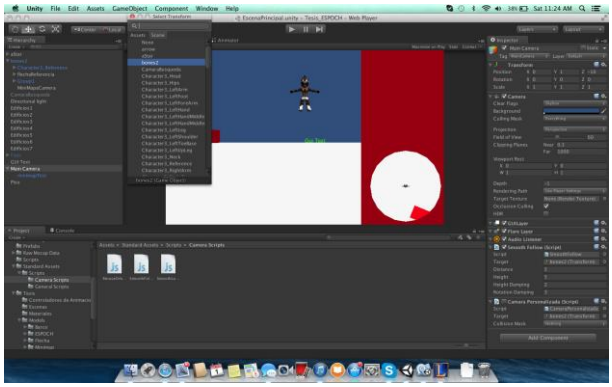


Figura 154. Señalar el personaje al que seguirá la cámara

6. En el resto de opciones se eligen valores a conveniencia, para esto ayudarse de la Vista de Juego dando clic en ► e ir seleccionando valores hasta encontrar los adecuados.

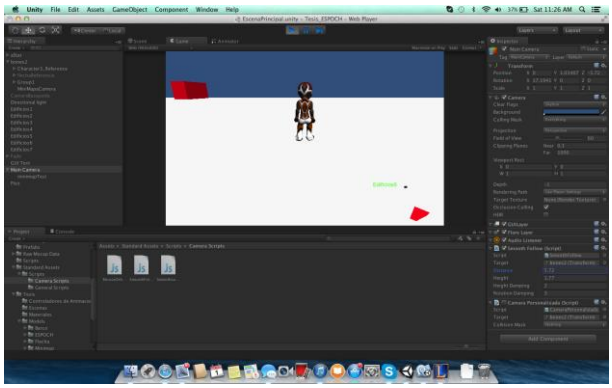


Figura 155. Configurar los valores de la cámara

5.3.16.Creación y programación de la Interfaz Gráfica de Usuario (GUI) para el menú principal

La Interfaz de Usuario es el primer contacto que tiene el usuario con la aplicación, es por eso que debe ser lo más intuitiva, sencilla y vistosa.

1. Importar todas las texturas necesarias para la GUI del menú principal como se vio en el punto 5.1.3.
2. Crear una nueva escena. Para esto en a la pestaña *File* seleccionar *New Scene*.

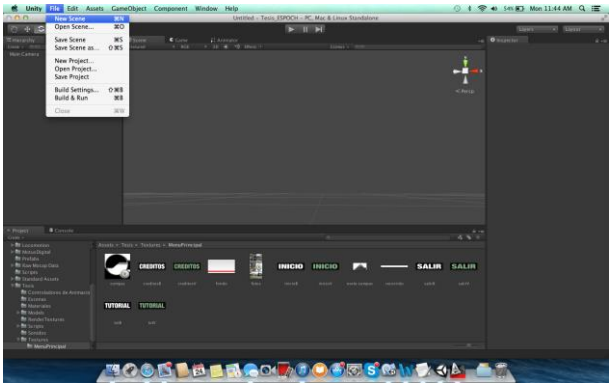


Figura 156. Crear una nueva escena

3. En la nueva escena crear un *Quad* para lo cual se debe ir a la pestaña *Game Object, Create Other* y seleccionar *Quad*.

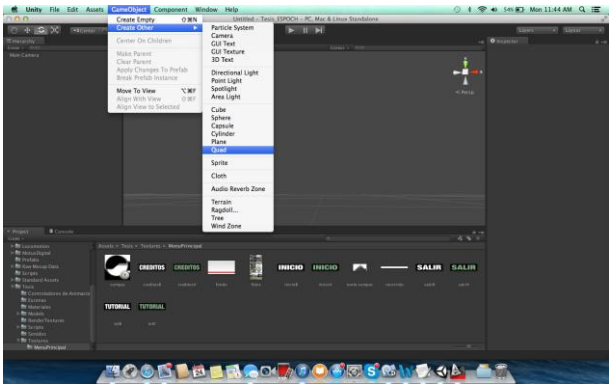


Figura 157. Crear un Quad

4. En la Vista de Jerarquía renombrar el Quad a *BG* (background).

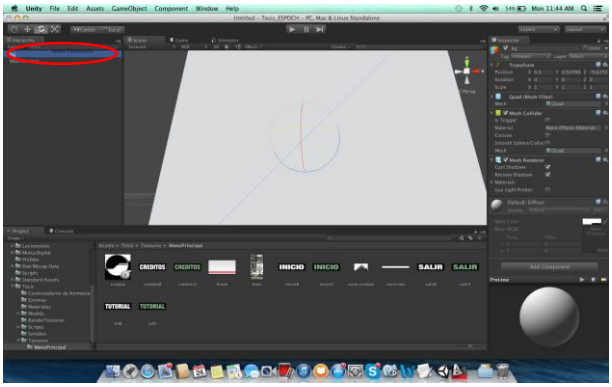


Figura 158. Renombrar el Quad a BG

5. En el Inspector cambiar el campo *Material* que se encuentra en la parte inferior a un tipo Texture dentro *Unlit* y asignar la textura que se va a utilizar de fondo.

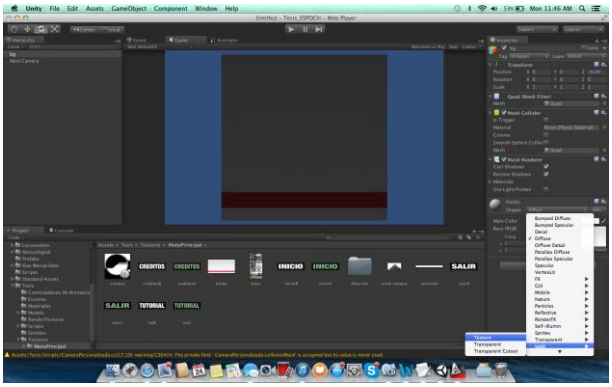


Figura 159.Asignar el material que se utilizará como fondo

6. En la *Vista de Jerarquía* seleccionar *MainCamara* y en el *Inspector* buscar la opción *Projection* y cambiar por Orthographic.

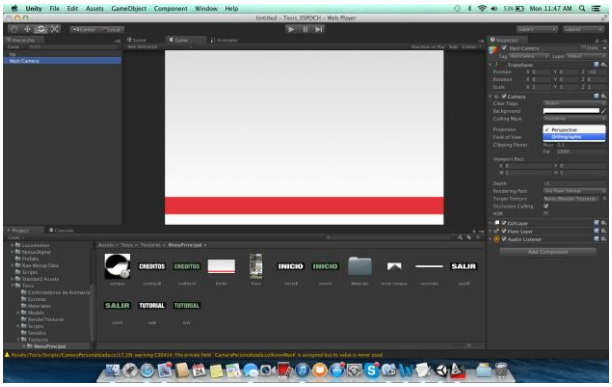


Figura 160. Cambiar la Proyección de la cámara Principal

7. Ajustar el valor de *Size* para que el fondo ocupe todo el espacio que se muestra en la cámara; para que sea más fácil configurar esto se puede cambiar a la Vista de Juego que muestra cómo va a quedar la Escena final, esta opción se encuentra junto al título de la Vista de Jerarquía.

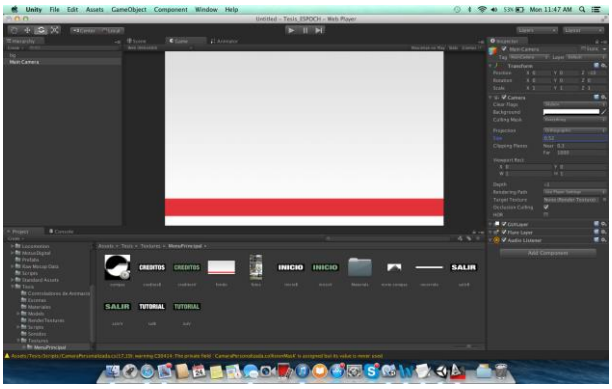


Figura 161. Ajustar el tamaño del fondo

8. Duplicar el fondo en la Vista de Jerarquía seleccionando *BG* y presionando ctrl + D y renombrar el nuevo objeto como *fotos*.

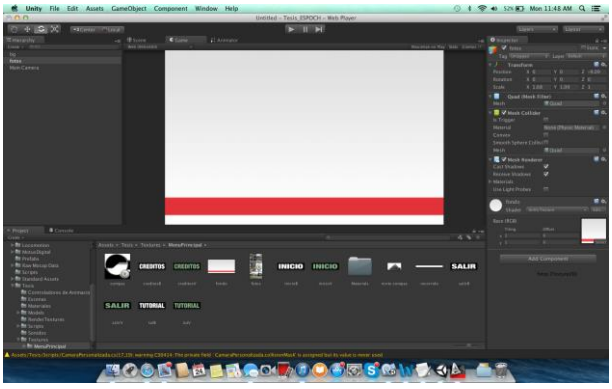


Figura 162. Duplicar BG para añadir el objeto fotos

9. Arrastrar la textura de fotos hacia la parte inferior del Inspector al botón *Add Component*, esto permitirá que se reemplace el material del elemento *fotos*.

   En el Inspector cambiar el campo Material que se encuentra en la parte inferior a un tipo Transparent dentro *Unlit.*

   Ajustar el valor de la posición y escala en el componente *Transform* del *Inspector* para ubicar el nuevo objeto en el lugar que ha sido planeado.

Figura 163. Configurar el objeto fotos

10. Para la creación de los botones de la interfaz duplicar el objeto *fotos* en la *Vista de Jerarquía* tantas veces como componentes se tenga en la interfaz, presionando nuevamente ctrl + D y el objeto seleccionado.
    Cambiar de nombre a cada objeto, asignarle el material respectivo a cada uno y posicionarle en el lugar deseado de la misma manera que se hizo en pasos anteriores.

Figura 164. Creación de los objetos botones

11. Duplicar por última vez un objeto y cambiarle de nombre ha *seleccionado*, este permitirá indicar que botón ha sido seleccionado y se lo ubicará en la parte superior agrandando su escala para que sea más visible. Cambiar la textura del objeto por una textura que sea completamente transparente.

Figura 165. Crear un objeto para mostrar el botón seleccionado

12. Guardar la escena, pra esto ir a la Pestaña *File* y seleccionar *Save Scene*.

Figura 166. Guardar la escena

13. Escoger el directorio en el cual se quiere guardar y asignarle un nombre a la escena. Clic en *Save*.

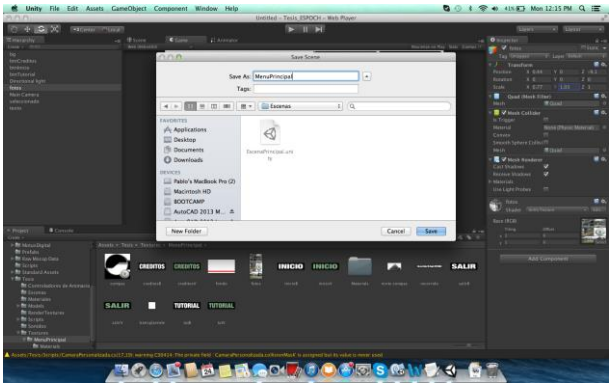


Figura 167. Seleccionar el directorio para guardar la escena

14. En la Vista de Proyecto en la carpeta “Scripts” crear un subdirectorio llamado “MenuPrincipal”. Dentro de él se crea un nuevo C# Script al que se lo llamará Boton.

Figura 168. Crear un Script llamado Botón

15. Doble clic en el Script y se abrirá una nueva ventana de Mono Develop. Declar un vector público de texturas con la siguiente línea de código public Texture[ ] texturas y añadimos el evento OnMouseOver, el cual imprimirá en la consola el nombre del objeto al cual se le añada este componente.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Buton : MonoBehaviour {
5      public Texture[] texturas;
6      // Use this for initialization
7      void Start () {
8
9      }
10
11     void OnMouseOver()
12     {
13         print (this.transform.name);
14     }
```

Figura 169. Crear variables para controlar la interfaz

16. Volver a Unity y en la Vista de Jerarquía seleccionar todos los objetos que son botones. En el Inspector buscar *Add Component*, seleccionar *Scripts* y escoger el Script que recién se creó llamado *Boton*.

Figura 170. Añadir el Script a los objetos Boton

17. Para probar que funcione correctamente el Script clic en ► en la Vista la Escena y se puede ver que al situar el mouse sobre uno de los botones en la consola se muestra el nombre del objeto al cual se añadió el Script.

Figura 171. Probar el Script realizado

18. Seleccionar uno por uno los botones en la *Vista de Jerarquía* y en el *Inspector* buscar el componente *Boton* que hace referencia al Script, expandir el campo de *Texturas* y en *Size* asignarle el valor de 2.

Figura 172. Permitir que cada botón tenga dos texturas

19. Hacia el campo Element 0 arrastrar desde la carpeta “Menu Principal” la textura que tiene el botón cuando se encuentra en un estado normal, al campo Element 1 arrastrar la textura que va a tomar el botón cuando el puntero del mouse se posicione sobre él.

Figura 173. Agregar las dos texturas a cada botón

20. Regresar al Script en Mono Develop y reemplazar el código que se encontraba en el evento OnMouseOver por la siguiente línea de código this.renderer.material.mainTexture = texturas [1]. Añadir además el evento OnMouseExit el cual contará con el siguiente código this.renderer.material.mainTexture = texturas [0], este le permitirá tomar el valor de la textura inicial cuando el puntero del mouse salga del botón.

```
using UnityEngine;
using System.Collections;

public class Buton : MonoBehaviour {
    public Texture[] texturas;

    // Use this for initialization
    void Start () {

    }

    void OnMouseEnter()
    {
        this.renderer.material.mainTexture = texturas [1];
    }

    void OnMouseExit()
    {
        this.renderer.material.mainTexture = texturas [0];
    }
    // Update is called once per frame
    void Update () {

    }
}
```

Figura 174. Código para método para que el botón tome las texturas de acuerdo a la interacción del mouse

21. Crear un nuevo script y llamarlo AnimacionUV, este script se utilizará para dar movimiento al título de la aplicación.

Figura 175. Crear un Script para dar movimiento al título de la aplicación

22. Doble clic sobre él para volver a MonoDevelop. Declarar una variable pública de tipo Vector2 llamada *velocidad* a la cual se le da un valor de 0.1 en X y 0 en Y; declarar también una variable privada a la que se llamará *offsetActual* la cual llevará el desplazamiento de las coordenadas UV.

    En el método Update sumar la velocidad al valor actual de la variable *offsetActual* y asignar el valor de esta variable al offset de la textura principal del objeto al que se le añadirá este script.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class AnimacionUV : MonoBehaviour {
5     public Vector2 velocidad = new Vector2(0.1f,0);
6     private Vector2 offsetActual = new Vector2();
7     // Use this for initialization
8     void Start () {
9
10    }
11
12    // Update is called once per frame
13    void Update () {
14        offsetActual += velocidad;
15        this.renderer.material.mainTextureOffset = offsetActual;
16    }
17 }
```

Figura 176. Script que permite la animación del texto

23. Volver a Unity y en la *Vista de Jerarquía* seleccionar el objeto al cual se le agregará el Script de *AnimacionUV*, en este caso es el que contiene el texto “Recorrido Virtual ESPOCH”. En el *Inspector* añadir el script AnimacionUV como se ha visto anteriormente.

Figura 177. Añadir el Script al texto

24. Para agregar movimiento al texto que se muestra al seleccionar un botón crear un nuevo Script de la misma manera que se ha venido realizando y nombrarlo AnimacionMovimiento.

    Ir a Mono Develop y en el nuevo Script creado declarar una variable privada de tipo Vector3 llamada *posicionOriginal*, una variable pública de tipo Vector3 con el nombre ve*l*ocidad a la cual le asignará el valor de -1 en X, 0 en Y y Z y declarar una última variable privada de tipo bool inicializada en false llamada *mover*.

    En el método Start guardar la posición inicial en la que se creará el objeto con posicionOriginal = this.transform.position.

    Crear un método público al que se nombrará ReiniciarPosicion el que regresará el objeto a su posición inicial, sin importar la posición en la que se encuentre actualmente con this.transform.position = posicionOriginal y cambiar el valor de la variable “mover” a true.

    Por último en el método Update preguntar, si la variable *mover* tiene el valor de verdadero a la posición de actual se le suma el vector *velocidad* multiplicado por Time.deltaTime.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class AnimacionMovimiento : MonoBehaviour {
5      Vector3 posicionOriginal;
6      public Vector3 velocidad = new Vector3(-1,0,0);
7      private bool mover = false;
8      // Use this for initialization
9      void Start () {
10         posicionOriginal = this.transform.position;
11     }
12
13     public void ReiniciarPosicion()
14     {
15         this.transform.position = posicionOriginal;
16         mover = true;
17     }
18     // Update is called once per frame
19     void Update () {
20         if (mover)
21         {
22             this.transform.position += velocidad * Time.deltaTime;
23         }
24     }
25 }
26
```

Figura 178. Código para agregar movimiento al texto seleccionado

25. Buscar en la Vista de Jerarquía el objeto *seleccionado* y añadir el script AnimacionMovimiento.

Figura 179. Añadir el Script creado al objeto que tendrá movimiento

26. Buscar el script Boton y en Mono Develop se le añade las variables públicas de tipo Texture llamada transparente, otra de tipo *GameObject* llamada seleccionado y una última de tipo String llamada siguienteEscena. Se le añade también 2 variables privadas una de tipo Renderer llamada materialSeleccionado y otra de tipo AnimacionMovimiento llamada animacionMovimiento.

```
public class Buton : MonoBehaviour {
    public Texture[] texturas;
    public Texture transparente;
    public GameObject seleccionado;

    public string siguienteEscena;

    private Renderer materialSeleccionado;
    private AnimacionMovimiento animacionMovimiento;
```

Figura 180. Código para crear variables para la interacción de los botones con las escenas

27. En el método Start se agrega una referencia a la variable “animacionMovimiento” con el componente “AnimacionMovimiento” por medio de la siguiente línea de código animacionMovimiento = seleccionado.GetComponent<AnimacionMovimiento>(); se referencia igualmente a la variable “materiaSeleccionado” de la siguiente manera materiaSeleccionado = seleccionado.renderer.

```
void Start () {
    materialSeleccionado = seleccionado.renderer;
    animacionMovimiento = seleccionado.GetComponent<AnimacionMovimiento> ();
}
```

Figura 181. Código para referenciar a la variable animacionMovimiento

28. Modificar el evento OnMouseEnter al que se añade la siguiente línea de código materialSeleccionado.material.mainTexture = texturas[0] para que tome la misma textura del botón seleccionado. Reiniciar la animación de movimiento con el código animacionMovimiento.ReiniciarPosicion().

    Añadir el evento OnMouseDown que se activará cuando se de clic sobre el botón, dentro del evento se insertará la siguiente línea de código Application.LoadLevel(siguienteEscena) con la cual se permitirá el movimiento entre escenas.

```
void OnMouseEnter()
{
    this.renderer.material.mainTexture = texturas [1];
    materialSeleccionado.material.mainTexture = texturas[0];
    animacionMovimiento.ReiniciarPosicion ();
}

void OnMouseDown()
{
    Application.LoadLevel (siguienteEscena);
}

void OnMouseExit()
{
    this.renderer.material.mainTexture = texturas [0];
}
```

Figura 182. Código para cargar la siguiente escena

29. Guardar cambios y regresar a Unity. Seleccionar todos los botones en la Vista de Jerarquía y en el Inspector buscar el componente Boton y dando clic en el campo Seleccionado se busca el objeto “seleccionado”. En el campo *Siguiente Escena* poner el nombre de la escena a la cual se llegará al presionar el botón; por último en el campo *Transparente* seleccionar la textura que es completamente transparente.

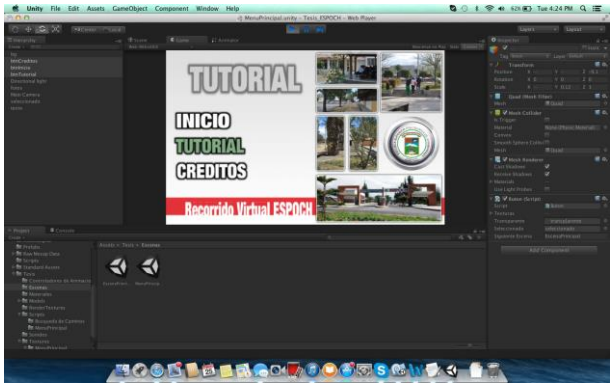


Figura 183. Añadir el Script de interacción a todos los botones

30. En la pestaña *File* seleccionar *Build Settings* lo que desplegará una nueva ventana.

Figura 184. Build Settings

31. Seleccionar el campo Web Player en Plataforma y presionar el botón *Add Current* para que se agregue la escena actual al ejecutable final.

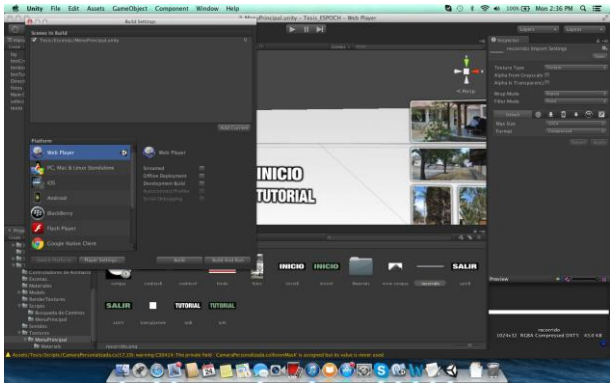


Figura 185. Añadir escenas a Build Settings

32. Volver a la escena anterior dando doble clic sobre la escena mencionada en la *Vista de Proyecto* y en la ventana *Build Settings* se presiona nuevamente *Add Current*.

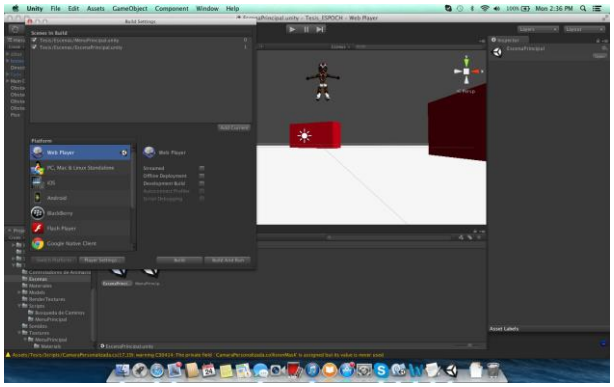


Figura 186. Añadir la escena principal a Build Settings

33. Regresar a la escena del menú y presionar ► en la Vista de Escena para probar que todos los componentes del menú funcionen correctamente.

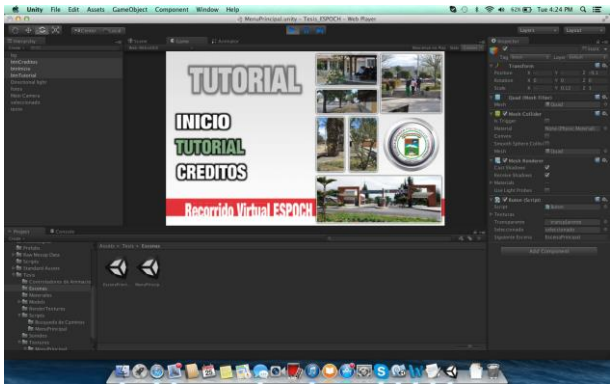


Figura 187. Probar el funcionamiento de los botones

5.3.17. Configuración del sonido

1. En la Vista de Jerarquía seleccionar el personaje, en la Pestaña *Component* seleccionar *Audio* y elegir *Audio Source*.

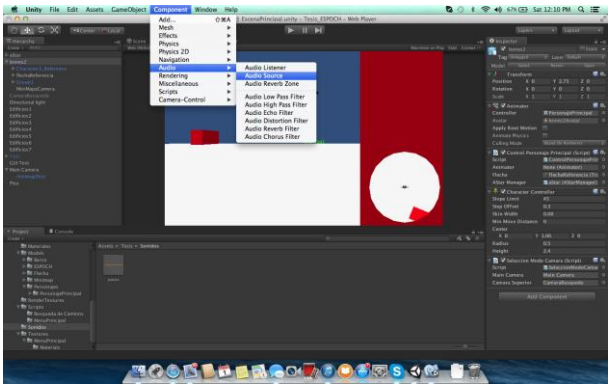


Figura 188. Agregar un AudioSource

2. En la Vista de Proyecto buscar la carpeta “Sonidos” donde se importaron los sonidos anteriormente. Buscar el sonido que se utilizará para pasos.

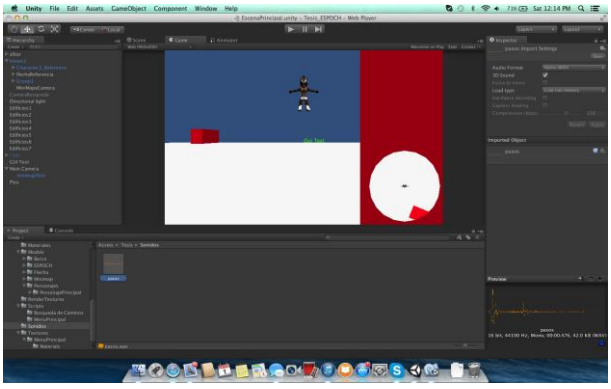


Figura 189. Seleccionar el sonido a utilizar

3. En la Vista de Jerarquía seleccionar el personaje y en el *Inspector* buscar el componente *Audio Source*. En el campo *Audio Clip* seleccionar el sonido encontrado anteriormente (en este caso pasos) y activar la opción *Loop* para una vez finalizada la reproducción del sonido se repita.

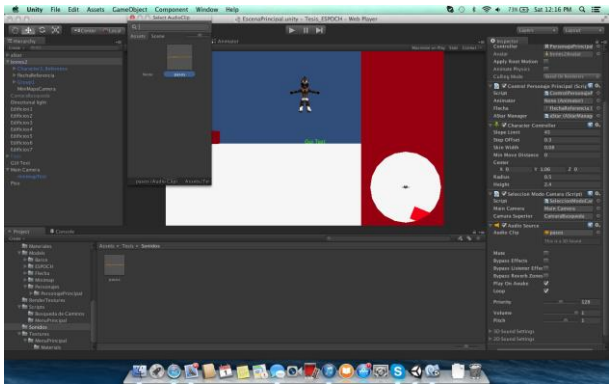


Figura 190. Agregar un Audio Clip al personaje

4. Dar clic en ► y se escuchará el sonido permanentemente, de esta forma cualquier sonido puede utilizarse como sonido de fondo.
5. En el caso de los pasos debe añadirse una interacción para que el sonido se reproduzca solamente cuando el personaje se encuentre en movimiento. Para esto en el Script ControlPersonajePrincipal se crea una variable privada de tipo AudioSource a la que se llamará sonidoPasos.

```
public class ControlPersonajePrincipal : MonoBehaviour {

    private AudioSource sonidoPasos;

    public Animator animator;
    public Transform flecha;
    public AStarManager aStarManager;

    private NodoEspacial nodoCercano;
    CharacterController controller;

    private float velocidad = 0;
    private float velocidadRotacion = 30;
    private float aceleracion = 1.5f;
```

Figura 191. Código para crear una variable que referencie al sonido en el Script del personaje

6. En el método Start asignar a la variable sonidoPasos una referencia al componente AudioSource que se agregó en el Inspector anteriormente.

```
// Use this for initialization
void Start () {
    animator = this.GetComponent<Animator> ();
    controller = this.GetComponent<CharacterController> ();
    sonidoPasos = this.GetComponent<AudioSource> ();
}
```

Figura 192. Código para referencias la variable pasos en el método Start

7. Ir al método Update y después del código que en él se encuentra, preguntar si la *velocidad* corresponde a los valores para que la animación que se ejecute sea caminar hacia adelante o hacia atrás (es decir si la *velocidad* está ente 0.3 y 1 o es menor que 0.1). Si la respuesta es afirmativa preguntar si el sonido no se está reproduciendo y reproducirlo.

```
if((velocidad>0.3f && velocidad <1) || (velocidad<-0.1f))
{
    if(!this.sonidoPasos.isPlaying)
    {
        this.sonidoPasos.Play();
    }
}
else
{
    this.sonidoPasos.Stop();
}
```

Figura 193. Código para reproducir el sonido con la animación de caminar

8. Regresar a Unity y dar clic en ► para probar que el sonido se ejecute solamente cuando el personaje camina.
9. En la *Vista de Jerarquía* seleccionar el personaje y añadir un componente *Audio Source* como se hizo anteriormente. En el Inspector ir a la opción *Audio Clip* del nuevo componente y seleccionar un sonido de correr. Activar la opción *Loop* y desactivar la opción *Play On Awake* de los *Audio Source* creados hasta el momento.

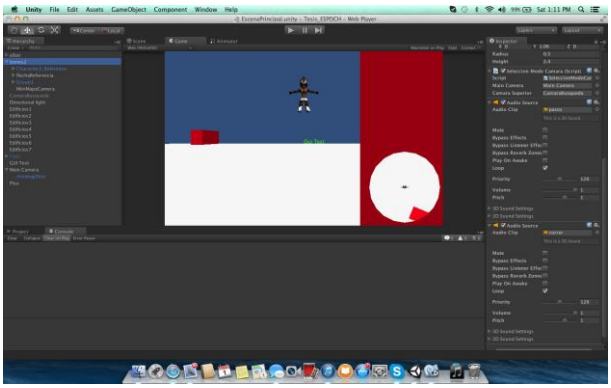


Figura 194. Agrergar un nuevo AudioSource

10. Regresar al Script ControlPersonajePrincipal y cambiar el tipo de la variable sonidoPasos de AudioSource a un vector de AudioSource.

```
public class ControlPersonajePrincipal : MonoBehaviour {

    private AudioSource[] sonidoPasos;

    public Animator animator;
    public Transform flecha;
    public AStarManager aStarManager;

    private NodoEspacial nodoCercano;
    CharacterController controller;

    private float velocidad = 0;
    private float velocidadRotacion = 30;
    private float aceleracion = 1.5f;
```

Figura 195. Código para cambiar el tipo de variable que referencia al sonido en el Script del personaje

11. En el método Start se cambia la función GetComponent que se le asigna a *sonidoPasos por* GetComponents para obtener un vector con la referencia a todos los componentes de tipo AudioSource añadidos a este objeto.

```
void Start () {
    animator = this.GetComponent<Animator> ();
    controller = this.GetComponent<CharacterController> ();
    sonidoPasos = this.GetComponents<AudioSource> ();
}
```

Figura 196. Código para obtener todos los componentes AudioSource

12. En el método Update anteriormente se controló que el sonido de pasos se reproduzca si la *velocidad* cumplía las condiciones de caminar, ahora se preguntará también si la *velocidad* cumple con las condiciones para que se reproduzca la animación correr y en este caso se reproducirá el sonido correr en lugar del sonido de pasos. Caso contrario se detienen todos los sonidos. Para esto se agregará una nueva condición.

```
if((velocidad>0.3f && velocidad <1) || (velocidad<-0.1f))
{
    if(!this.sonidoPasos[0].isPlaying)
    {
        this.sonidoPasos[0].Play();
    }
}
else if(velocidad>1)
{
    this.sonidoPasos[0].Stop();
    if(!this.sonidoPasos[1].isPlaying)
    {
        this.sonidoPasos[1].Play();
    }
}
else
{
    foreach(AudioSource s in sonidoPasos)
    {
        s.Stop();
    }
}
```

Figura 197. Código para reproducir un sonido diferente de acuerdo a cada animación

13. Guardar cambios, volver a Unity y dar clic en ► para comprobar que el sonido que se reproduzca sea el adecuado a la animación de correr y caminar respectivamente.

5.3.18. Configuración de Culling

1. En la Pestaña *Window* escoger *Occlusion Culling*.

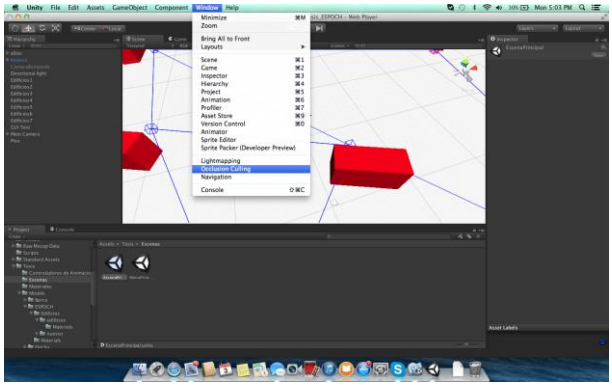


Figura 198. Occlusion Culling

2. En el *Inspector* se agregará automáticamente una nueva pestaña llamada *Occlusion*. En esta pestaña seleccionar *Renderers* en la opción *Object*.

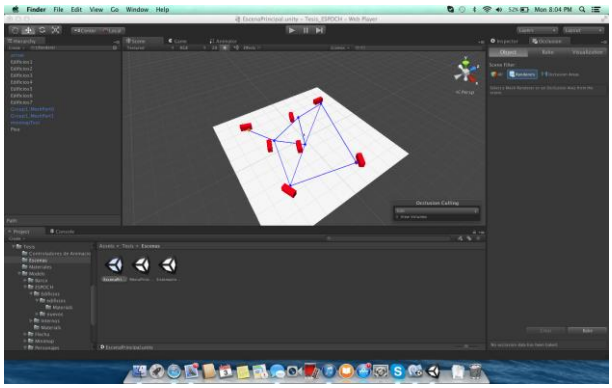


Figura 199. Configuración de Occlusion

3. En la *Vista de Jerarquía* seleccionar todos los objetos que permanecerán estáticos y no tienen transparencia. Ir al *Inspector* y seleccionar la opción *Static*.

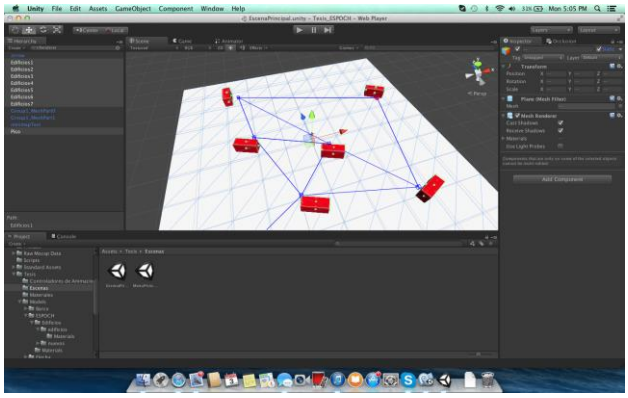


Figura 200. Volver estáticos a los objetos de la escena

4. Volver a la Pestaña *Occlusion* junto al *Inspector* y escoger la opción *Bake*. Asignar los valores necesarios de acuerdo a la escala del Mundo Virtual y dar clic en el botón *Bake*.

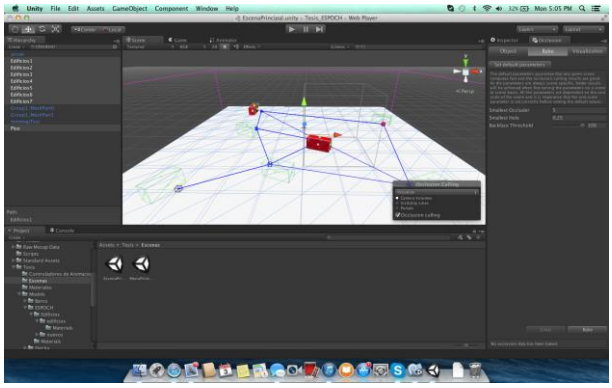


Figura 201. Asignar al Mundo Virtual la opción Bake

5. Se podrá visualizar las regiones en las que el motor subdivide la escena para la optimización en el momento del renderizado.

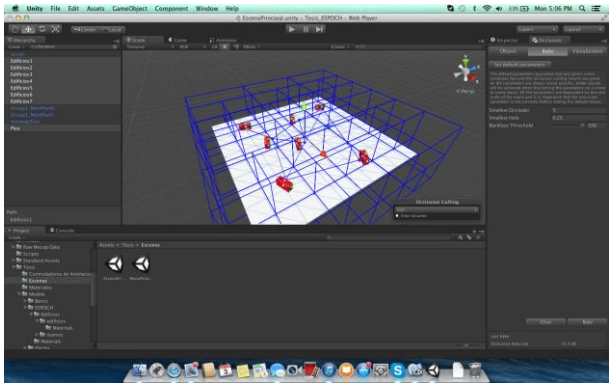


Figura 202. Mundo Virtual dividido en Escenas de Culling

6. Probar el funcionamiento dando clic en ► y en la Vista de Juego se puede activar la opción Stats y se puede ver que la escena funciona con un mayor número de fps.

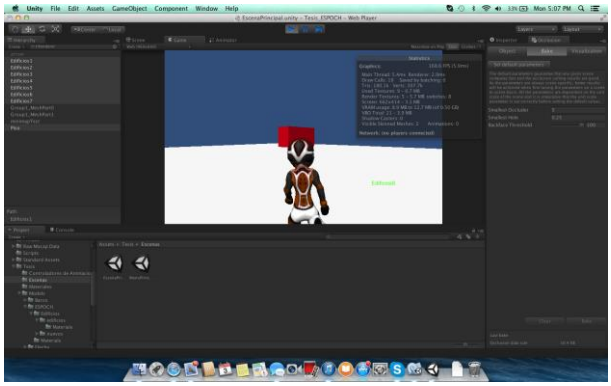


Figura 203. Escena con Occlusion Culling

5.3.19. Generar la aplicación

1. En la Pestaña *File* buscar la opción *Build Settings*.

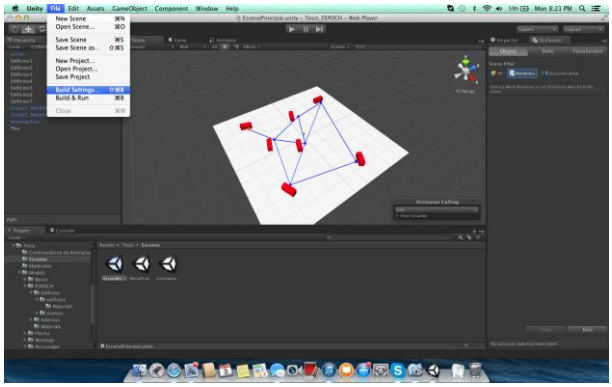


Figura 204. Buscar la opción Build Settings

2. Se mostrará una nueva ventana llamada *Build Settings*. En la opción *Plataform* escoger *Web Player* y dar clic en *Player Settings* para configurar en el *Inspector* las opciones finales.

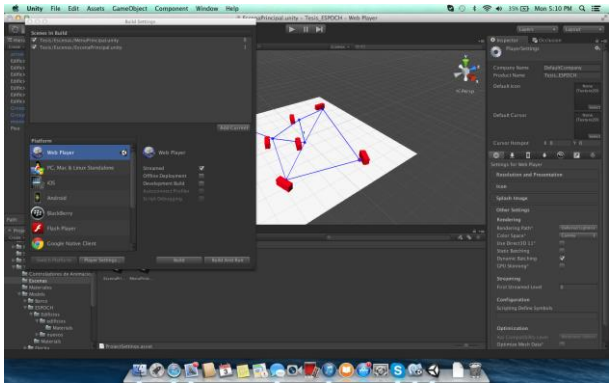


Figura 205.Ventana de Build Settings

3. Dar clic en la opción *Build* y seleccionar la ruta del directorio y el nombre para el proyecto final. Clic en *Save* para guardar cambios.

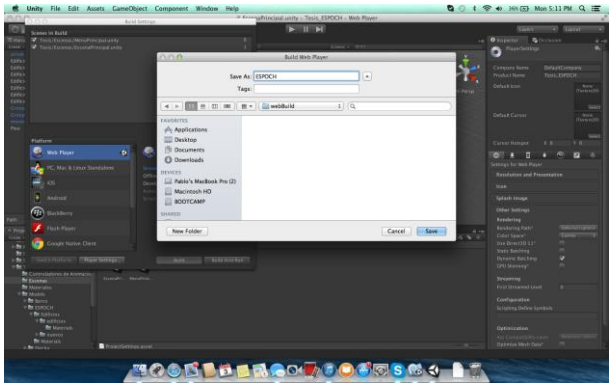


Figura 206. Ruta para guardar el proyecto

4. El motor automáticamente empezará a generar el archivo de la aplicación que funcionará en la Web.

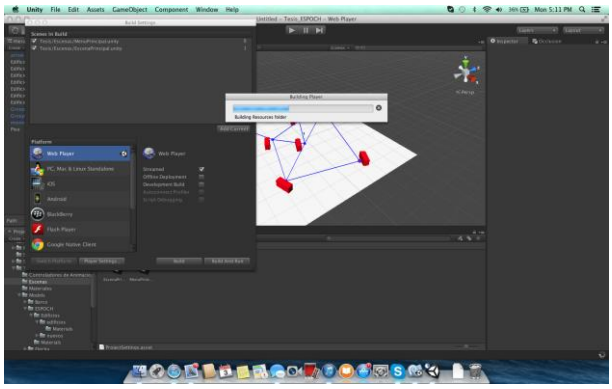


Figura 207. Generación automática de la aplicación

5. Al Abrir desde un navegador el documento html generado, la aplicación empezará a funcionar correctamente.

Figura 208. Aplicación en funcionamiento

CAPITULO VI

COMPROBACIÓN Y APLICACIÓN DE LA METODOLOGÍA AL CASO PRACTICO ESPOCH

Una vez que se ha especificado la metodología genérica para utilizarse en la creación de Mundos Virtuales, se procederá a comprobarla; para esto se definirán parámetros de evaluación y las variables respectivas para su comprobación.

Una vez probada la metodología se aplicarla en la creación del Campus Virtual de la ESPOCH. Posteriormente se evaluarán los resultados obtenidos en función a los parámetros de evaluación planteados en este capítulo.

6.1. Comprobación de la metodología

6.1.1. Definición de parámetros y variables de evaluación

Para evaluar el correcto funcionamiento de la metodología desarrollada se definirán parámetros para comprobar si la metodología planteada funciona de la manera esperada.

Los parámetros que se utilizarán para evaluar el correcto funcionamiento de un Mundo Virtual son:

6.1.1.1. Cantidad de Texturas

Para comprobar la cantidad de texturas se necesitará medir la memoria que utiliza la aplicación y el tamaño del ejecutable en función a cuantas texturas se utilizarán en un proyecto. La variable a utilizar será ***T*** que hará referencia a con cuantas texturas se hará la prueba.

6.1.1.2. Cantidad de Vértices

Para la comprobación de la cantidad de luces en el proyecto el valor a medir será el de los fps (frames por segundo; para esto se utilizará la variable ***V*** que hace referencia a la cantidad de vértices que se utilizan en el proyecto y permitirá el cálculo de los fps a los que correrá la aplicación.

6.1.1.3. Cantidad de Luces

La cantidad de luces, para su comprobación, medirá el valor de los fps a los que correrá la aplicación de acuerdo a la variación de ***L***, que referenciará a la cantidad de luces con las que contará la aplicación.

6.1.1.4. Tamaño de las Texturas

Una textura de un modelo puede tener diferentes tamaños. Para comprobar los fps a los que correrá la aplicación se variará ***Tt*** que hace referencia al tamaño de las texturas.

6.1.1.5. Complejidad del Material

El material utilizado en una aplicación puede tener diferente complejidad de acuerdo a shader que se utilice; ya que para el cálculo que se realice internamente en el motor se tomará en cuenta la cantidad de texturas, las luces, etc. Para comprobar que la complejidad del material utilizada sea la correcta se medirán los fps en función a la variable ***M*** que hará referencia a la complejidad usada en el material.

6.1.1.6. Optimizaciones (Culling de ángulo de Vista)

Un motor de video juegos optimizará la aplicación procesando solamente lo que se encuentre dentro del foco de vista. Para esto se variará la distancia que capta la cámara ***D*** y se medirá los fps a los que funciona el proyecto de acuerdo a estos valores.

6.1.2. Pruebas individuales por parámetro

Una vez definidas las variables a utilizarse en la comprobación de la metodología, se realizaron las pruebas de acuerdo a cada parámetro.

Para estas pruebas se utilizó la herramienta Profiler que viene incluida en el motor Unity 3D. Se utilizaron valores variables (***V***) y contantes(***C***), esta nomenclatura se antepondrá a la letra asignada a cada parámetro de acuerdo a la fórmula a utilizarse. En las unidades se utilizará la letra ***k*** para unidades de mil y para millón se utilizará la letra ***M***.

6.1.2.1. Cantidad de Texturas

Para medir la cantidad de memoria utilizada se utilizará la fórmula [6.1]

$$x = VT + CV + CL + CTt + CM + CD \qquad [6.1]$$

y los valores a utilizar serán:

- $VT_1 =$ 2
- $VT_2 =$ 10
- $VT_3 =$ 45
- $CV =$ 1.5M
- $CL = 2$
- $CTt =$1024x1024 (comprimida)
- $CM =$ Parallax Specular
- $CD =$ 1000

TABLA VI.I. COMPARACIÓN CANTIDAD DE TEXTURAS

	VT_1	VT_2	VT_3
Cantidad de Texturas	2	10	45
Uso de memoria	16.0 MB	22.7 MB	44.8 MB
Tamaño Ejecutable	1.89 MB	3.73 MB	11.9 MB

Fuente: Propia

Una vez analizados los datos, en la Tabla VI.I., se puede ver que a menor cantidad texturas el uso de memoria y el tamaño del ejecutable son menores, lo que implica que la aplicación funcionará más rápido.

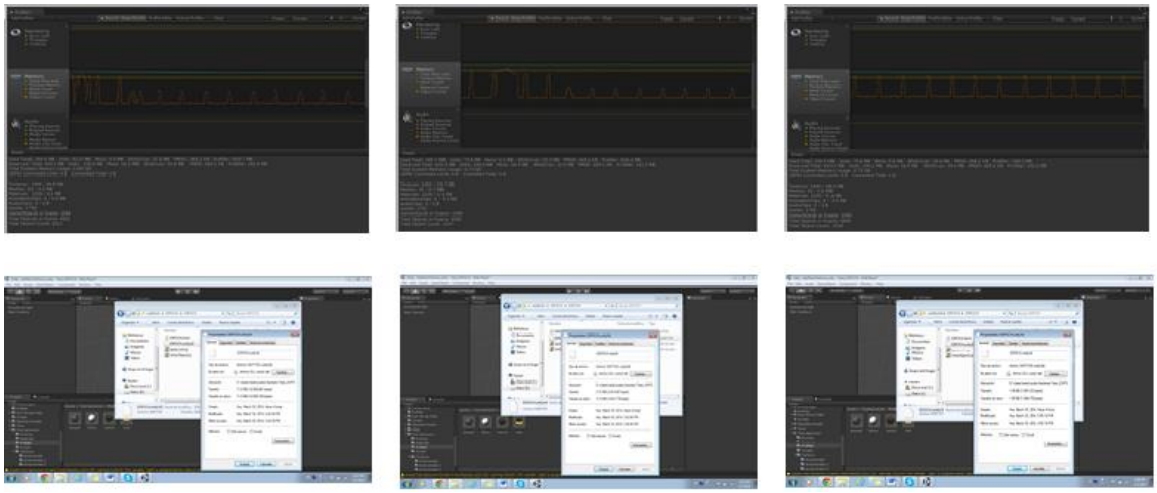


Figura 209. Pruebas cantidad de texturas

6.1.2.2. Cantidad de Vértices

Para medir los fps que se utilizan en la aplicación de acuerdo a la cantidad de vértices se utilizará la fórmula [6.2]

$$x = CT + VV + CL + CTt + CM + CD \qquad [6.2]$$

y los valores a utilizar serán:

- $VV_1 =$ 997.5k
- $VV_2 =$ 3.0M
- $VV_3 =$ 8.0M

- CT =45
- CL =2
- CTt = 1024x1024 (comprimidas)
- CM = Parallax Specular
- CD = 1000

TABLA VI. II. COMPARACIÓN CANTIDAD DE VÉRTICES

	VV_1	VV_2	VV_3
Cantidad de Vértices	997.5k	3.0M	8.0M
Fps	60-68 (66.5)	20-24 (23.3)	8-10 (9.2)

Fuente: Propia

Con los datos analizados en la Tabla VI.II., se puede ver que con una cantidad menor de vértices la aplicación funcionará a una cantidad superior de fps; notándose así que la aplicación será más fluida mientras menos vértices tengan los objetos de la escena.

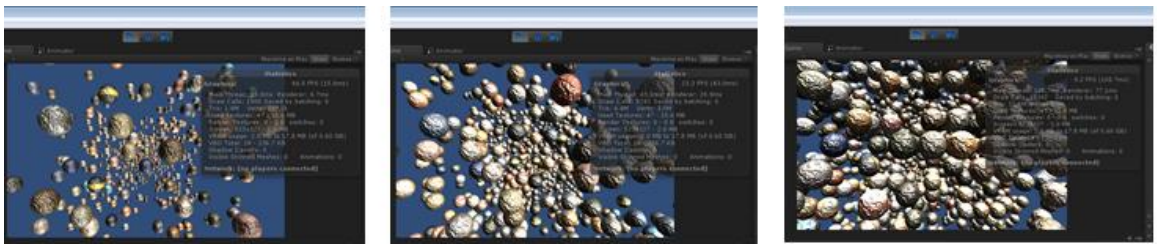


Figura 210. Pruebas cantidad de vértices

6.1.2.3. Cantidad de Luces

$$x = CT + CV + VL + CTt + CM + CD \qquad [6.3]$$

y los valores a utilizar serán:

- VL_1 = 2
- VL_2 = 10
- VL_3 = 200
- CV = 1.5M
- CT = 45

- $CTt =$ 1024x1024 (comprimidas)
- $CM =$ Parallax Specular
- $CD =$ 1000

TABLA VI. III. COMPARACIÓN CANTIDAD DE LUCES

	VL_1	VL_2	VL_3
Cantidad de Luces	2	10	200
Fps	63-67(66.0)	44-46 (45.3)	17-19 (18.4)

Fuente: Propia

Con los datos analizados. En la Tabla VI.III., se muestra que a menor cantidad de luces utilizadas la aplicación correrá a mayor cantidad de fps; por lo tanto se verá más fluida.

Figura 211. Pruebas cantidad de luces

6.1.2.4. Tamaño de las Texturas

$$x = CT + CV + CL + VTt + CM + CD \quad [6.4]$$

y los valores a utilizar serán:

- $VTt_1 =$ 4096x4096 (16bits)
- $VTt_2 =$ 1024x1024 (comprimido)
- $VTt_3 =$ 256x256 (comprimido)
- $CV =$1.5M
- $CL =$2
- $CT =$15
- $CM =$ Parallax Specular
- $CD =$ 1000

TABLA VI. IV. COMPARACIÓN TAMAÑO DE TEXTURAS

	VTt_1	VTt_2	VTt_3
Tamaño de las texturas	4096x4096 (16 bits)	1024x1024 (comprimido)	256x256 (comprimido)
Cantidad de memoria	0.63 GB	20.1 MB	10.7 MB
Tamaño del ejecutable	Error “sin memoria” construyendo la aplicación	4.73 MB	1.16 MB

Fuente: Propia

En la Tabla VI.IV., con los datos de prueba se ve que mientras más pequeña es la textura la cantidad de memoria y el ejecutable son más pequeños y más manejables; esto permitirá que la aplicación sea más liviana, más rápida y de fácil acceso.

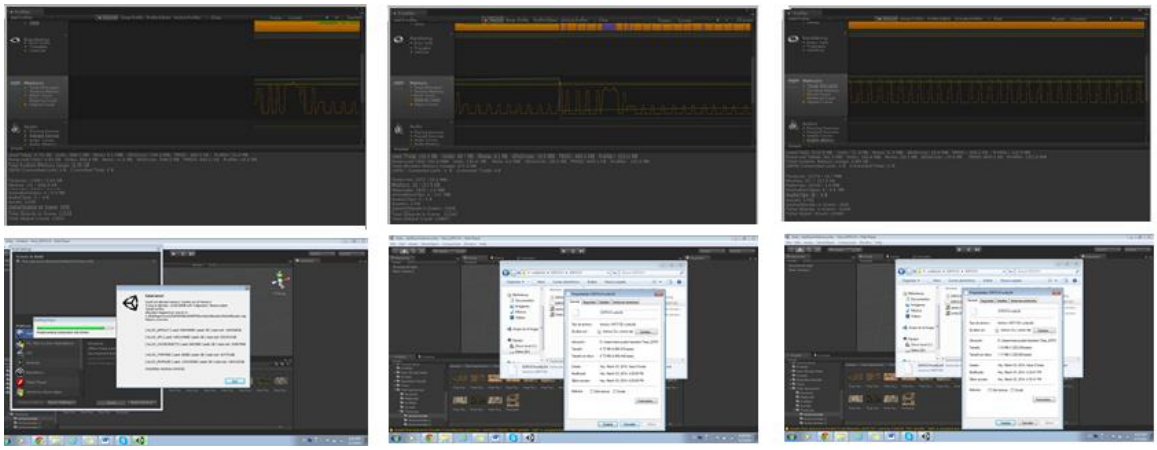


Figura 212. Pruebas tamaño de texturas

6.1.2.5. Complejidad del Material

$$x = CT + CV + CL + CTt + VM + CD \qquad [6.5]$$

y los valores a utilizar serán:

- $VM_1 =$ Unlit
- $VM_2 =$ Specular (Phong)
- $VM_3 =$ Vertex Lit (Particle)

- $VM_4 =$ Parallax Specular
- $CV =$ 1.5M
- $CL = 2$
- $CTt =$ 1024x1024 (comprimidas)
- $CT = 45$
- $CD = 1000$

TABLA VI. V. COMPARACIÓN COMPLEJIDAD DEL MATERIAL

	VM_1	VM_2	VM_3	VM_4
Complejidad del Material	Unlit	Specular (Phong)	Vertex Lit (Particle)	Parallax Specular
fps	63-67 (66.1)	45-48 (47.5)	51-55(53.4)	40-43(42.0)

Fuente: Propia

Con los datos analizados en la Tabla VI.V., se ve que mientras más simple es el shader que se utilice en el material, más rápida y más fluida será la aplicación pero será menos realista.

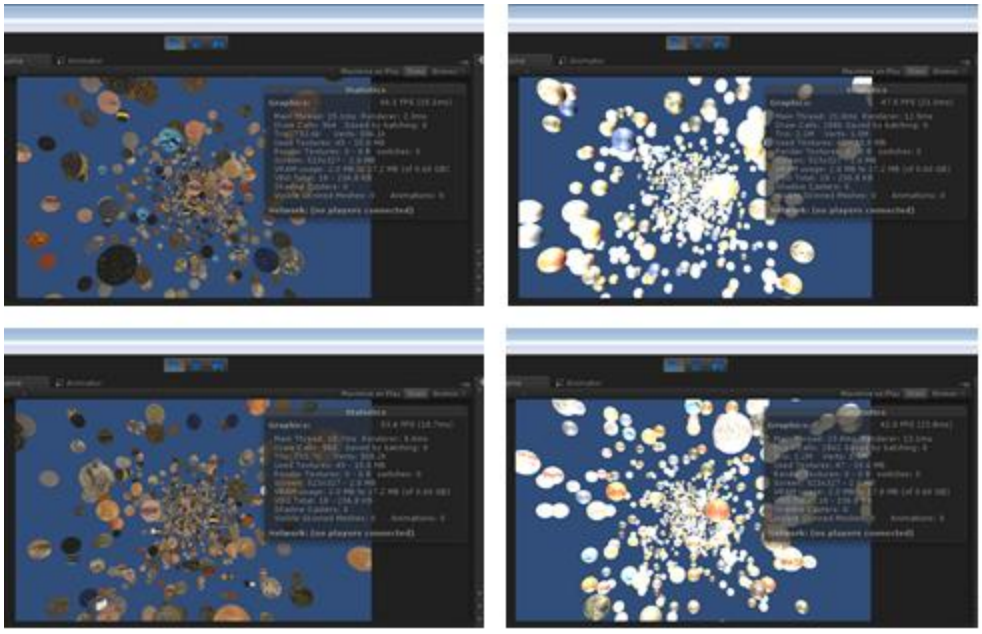


Figura 213. Pruebas complejidad del material

6.1.2.6. Optimizaciones (Culling de ángulo de Vista)

$$x = CT + CV + CL + CTt + CM + VD \quad [6.6]$$

y los valores a utilizar serán:

- $VD_1 =$ 500
- $VD_2 =$ 50
- $VD_3 =$ 10
- CV =12.1M
- $CL = 2$
- CTt =1024x1024 (comprimidas)
- $CM =$ Parallax Specular
- $CT = 45$

TABLA VI. VI. COMPARACIÓN DISTANCIA DE LA CÁMARA

	VD_1	VD_2	VD_3
Distancia de la cámara	500	50	10
fps	6-7.2 (6.3)	15-17 (16.0)	66-68 (67.9)

Fuente: Propia

De acuerdo a los datos analizados en la Tabla VI.VI., mientras menor sea la distancia que cubre el ángulo de vista de la cámara, los fps de la aplicación serán superiores. Con esto se nota que los elementos que se encuentren fuera del ángulo de vista no serán procesados, dejando así que la aplicación fluya de una mejor manera.

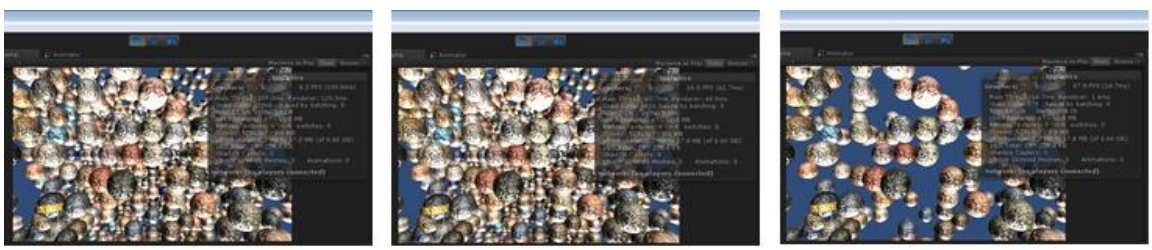


Figura 214. Pruebas distancia de la cámara

6.1.2.7. Características de la máquina para las pruebas

Para realizar las pruebas del proyecto se utilizó una máquina con las siguientes características:

TABLA VI. VII. CARACTERÍSTICAS DE LA MAQUINA DE PRUEBA

Sistema Operativo	Windows 7 Professional
Procesador	Intel Core i5-3210M
Velocidad del Procesador	2.50 GHz
Cantidad de Memoria	4 GB
Disco Duro	1 TB

Fuente: Propia

6.2. Aplicación de la metodología al caso práctico ESPOCH

La metodología genérica desarrollada en capítulos anteriores se aplicó a la creación de un Campus Virtual de la ESPOCH. El Campus Virtual incluye los edificios más representativos y permite al usuario caminar entre ellos; permite además buscar rutas entre un edificio y otro. A continuación se detallará el desarrollo de la metodología:

Se importaron los modelos necesarios para creación del Campus Virtual como son: personaje, terreno, edificios y demás assets a utilizarse.

Figura 215. Importación de modelos al Campus Virtual

Se ajustaron los parámetros de materiales y texturas importados para su posterior programación.

Figura 216. Ajustes de parámetros de materiales y texturas

Se continuó con la creación de estados y transiciones para la animación de “caminar” y “correr” del personaje.

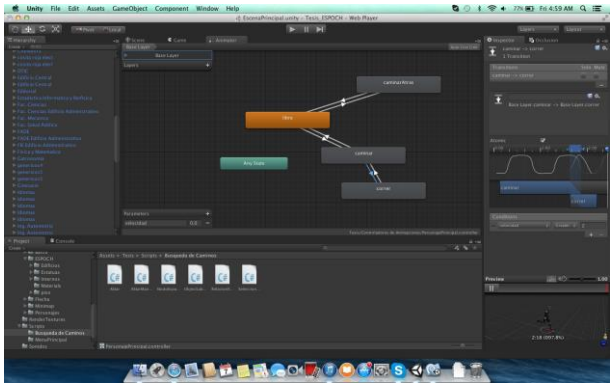


Figura 217. Creación de estados y transiciones del personaje

Se agregó interacción con el teclado y con la física al personaje. En este punto el personaje es controlado por el teclado y el sistema de colisiones entre los edificios y el personaje se encuentra funcionando correctamente.

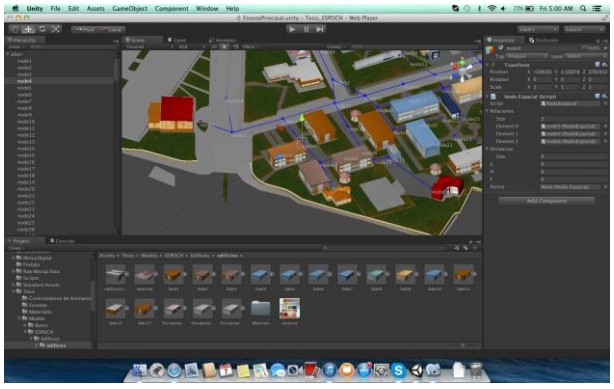


Figura 218. Creación de colisiones para los edificios

Se crea el mini mapa para la vista superior del Campus Virtual, el cual ayudará a la búsqueda de nodos que se implementará posteriormente.

Figura 219. Mini mapa del Campus Virtual

Se crearon un total de 146 nodos correspondientes a cada edificación y los caminos existentes entre todas ellas. Se implementó el algoritmo de búsqueda y se asignaron los nodos más cercanos a cada edificio como referencia a éstos.

Figura 220. Creación de nodos y asignación a los Edificios

Se programaron los componentes visuales que permiten que al dar clic en un edificio se pueda mostrar su nombre y la flecha que guiará al personaje de su ubicación actual al edificio al que desee llegar.

Figura 221. Componentes visuales del Campus Virtual

Como siguiente punto se programó la interfaz de usuario, sus botones y animaciones que permitirá entrar al Campus Virtual.

Figura 222. Interfaz de Usuario del Campus Virtual

Se agregaron sonidos y se configuró el Culling de la aplicación para que se realicen las optimizaciones.

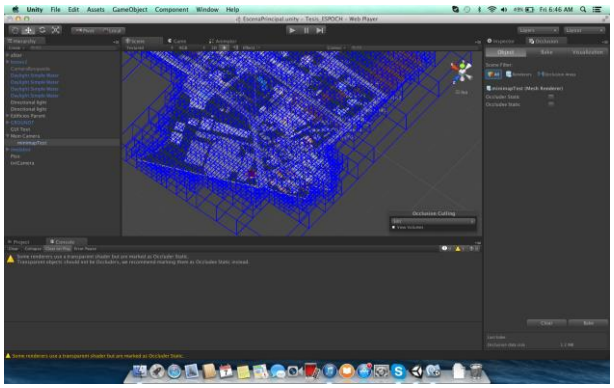


Figura 223. Culling de Oclusión

Como siguiente paso se generó la aplicación que será subida a la Web.

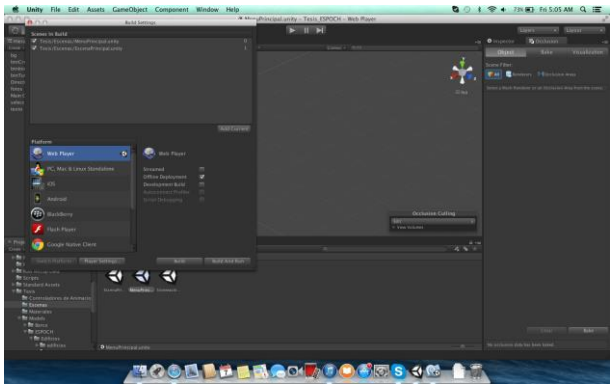


Figura 224. Generar el proyecto pasa publicar en la Web

Finalmente se verificará el funcionamiento de la aplicación final accediendo desde un navegador Web.

Figura 225. Campus Vitual ESPOCH

6.3. Comprobación de la hipótesis

6.3.1. Obtención de resultados del Campus Virtual de la ESPOCH

Una vez finalizada la aplicación del Campus Virtual de la ESPOCH se realizaron análisis, obteniendo los resultados que se muestran en la Tabla VI.VIII y en las Figuras de la 226 a la 228.

TABLA VI.VIII. DATOS OBTENIDOS DEL CAMPUS VIRTUAL ESPOCH

	RESULTADO	OBSERVACIONES
Número de Texturas	4	Dentro del rango de aceptabilidad en las pruebas
Cantidad de Vértices	748.3K	Menor a los valores de prueba utilizados, lo que implica mejor rendimiento de la aplicación
Cantidad de Luces	2	Dentro del rango de aceptabilidad en las priebas
Tamaño de las texturas	1024x1024(comprimido)	Dentro del rango de aceptabilidad en las pruebas
Complejidad del Material	Cerramiento: Transparent Diffuse	Se utilizaron diferentes tipos de materiales para obtener detalles de acuerdo a las necesidades de cada modelo
	Edificios: Diffuse	
	Personaje: Bumpped	
	Fuentes: Water	
Distancia de la cámara	1000	Aunque el valor de la cámara es superior a los valores de prueba, la aplicación funciona a más fps debido a que el Culling de Oclusión optimiza la aplicación con datos pre-calculados
fps	62 fps	La cantidad de fps a los que funciona la aplicación es mayor a la esperada. Esto implica que la aplicación se verá más fluida y que la metodología aplicada fue óptima.
Memoria Utilizada	43 MB	Se utiliza más memoria debido a que el Campus Virtual cuenta con más objetos que el test inicial.
Tamaño del Ejecutable	7.2 MB	Este valor permite que con un ancho de banda de 1 Mb el tiempo de descarga sea 1 min.

Fuente: Propia

Los valores que se muestran en las TABLA VI.VIII se los puede obtener en Unity 3D con la herramienta Profiler. Para esto en la pestaña *Window* seleccionar *Profiler (Ver Figura 226).* Se obtienen más datos para el análisis en Unity dando clic en *Stats* que se encuentra en la parte superior derecha de la Vista de Juego (Ver Figura 227). Para obtener el tamaño del archivo se buscan las opciones del archivo en el explorador (Ver Figura 228)

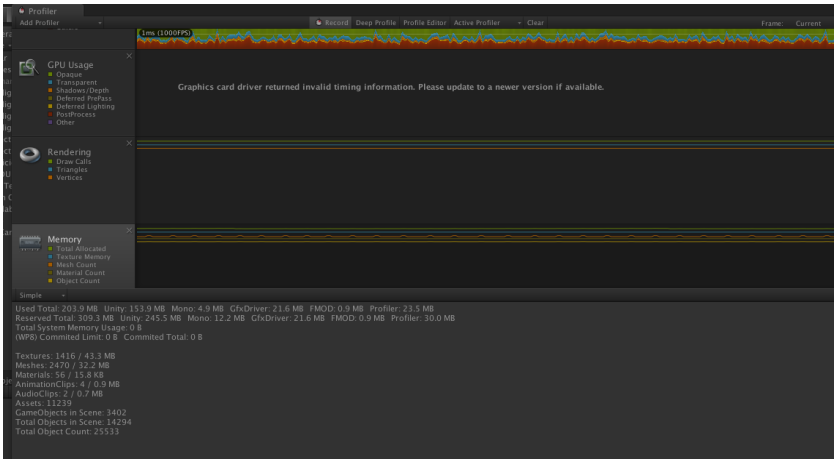


Figura 226. Herramienta Profiler

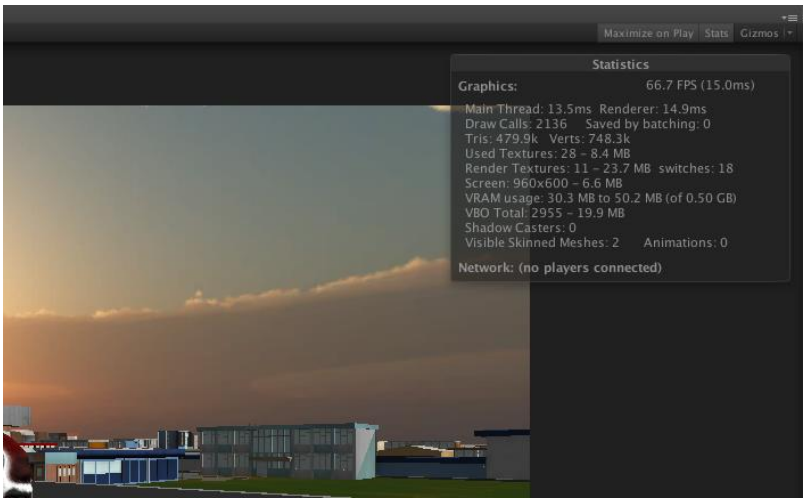


Figura 227. Herramienta Stats

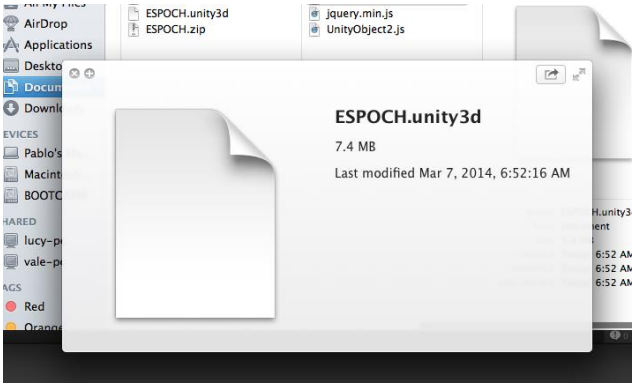


Figura 228. Propiedades del archivo

6.3.2. Comprobación de la metodología

- Todo proceso sin orden lleva a pérdida de recursos, principalmente de tiempo.
- La metodología propuesta reduce el tiempo de la programación de un mundo virtual ya que considera actividades en paralelo y verificación en cada fase; esto implica que en las pruebas finales no se requerirá regresar a fases anteriores para modificaciones, o en caso de ser necesarias no implicarán un retraso significativo en los tiempos de desarrollo de la aplicación.

A continuación se presenta un análisis del proceso de aplicación de la metodología al Campus Virtual de la ESPOCH, se describen los tiempos empleados en cada fase (Tabla VI.IX) y el tiempo general utilizado en la aplicación utilizando la metodología; así como un tiempo que se utilizaría para generar la misma aplicación sin la utilización de metodología planteada (Tabla VI.IX).

TABLA VI. IX. TIEMPOS EMPLEADOS POR FASE

FASE DESARROLLADA	TIEMPO EMPLEADO
Creación de un Nuevo proyecto	0.20 horas
Importación de los modelos creados en la herramienta Autodesk Maya 2014 hacia el motor de video juegos Unity 3D	0.50 horas
Importación de texturas creadas en la herramienta Photoshop hacia el motor de video juegos Unity 3D	0.33 horas
Importación de animaciones creadas en la herramienta Autodesk Maya 2014 hacia el motor de video juegos Unity 3D	0.33 horas
Descarga e importación de sonidos desde el Asset Store	0.25 horas
Ajuste de materiales, texturas, sonidos y animaciones	24 horas
Creación y programación de transiciones y estados para configurar las animaciones	5 horas
Programación de la interacción del personaje con el teclado	5 horas
Programación de la interacción del personaje con la física del Campus Virtual	40 horas
Creación del Mini Mapa que mostrará la búsqueda de caminos	8 horas
Implementación del algoritmo de búsqueda de caminos y asignación de nombres a los edificios principales	40 horas
Programación del personaje para la búsqueda de caminos	24 horas
Programación de componentes visuales	16 horas
Creación de la Interfaz de Usuario principal de la aplicación	8 horas
Programación del sonido Diegético Interactivo del personaje	4 horas
Configuración de la optimización por Culling	3 horas
Generación de la aplicación	1 hora
Pruebas finales	3 horas

Fuente: Propia

Utilizando la metodología planteada ciertas actividades se realizaron en paralelo, mientras que sin utilizar la metodología el desarrollo de la aplicación es secuencial y desordenado lo que implica un mayor tiempo de desarrollo como se puede ver en la Tabla VI.X

TABLA VI. X. TIEMPO DE USO DE LA METODOLOGÍA VS. TIEMPO SIN USO DE LA METODOLOGÍA

USANDO LA METODOLOGÍA		**SIN USAR LA METODOLOGÍA**	
En actividades en paralelo el tiempo tomado para la estimación es el tiempo de la actividad más larga. (Ver Figura 229)		En actividades secuenciales se suman los tiempos empleados en cada fase lo que implica un tiempo más largo en el desarrollo de la aplicación. (Ver Figura 230)	
Se cuenta con puntos de verificación en cada fase, por lo que para avanzar a la siguiente ya no existirán errores. Con esto se consigue que al llegar al punto de verificación final no se presenten cambios significativos.		Al no contar con puntos de verificación en cada fase, una vez que se llega a la verificación final si existiesen errores se debe regresar a puntos anteriores, esto implica pérdida de tiempo y que use más tiempo del estimado.	
Total Horas empleadas: **126. 61 horas**	Total Días de trabajo: (8 horas) **15 días**	Total Horas empleadas: **185.61 horas**	Total Días de trabajo: (8 horas) **24 días**

Fuente: Propia

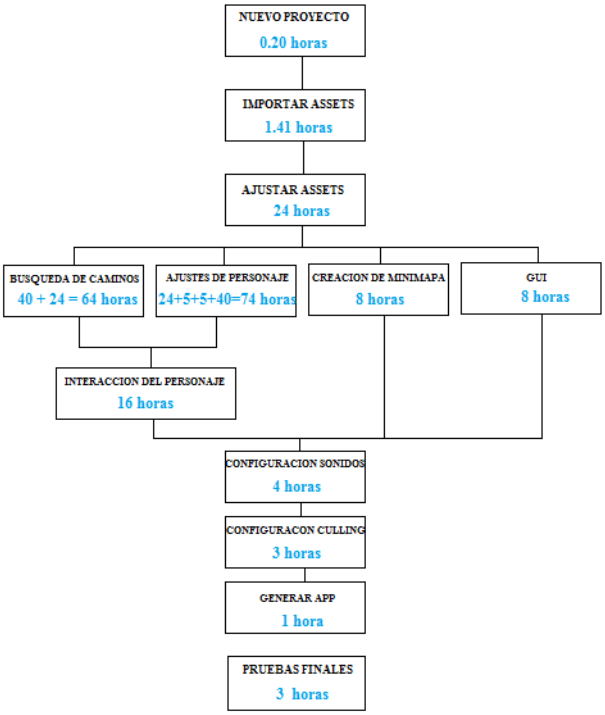


Figura 229. Tiempos de desarrollo empleados utilizando la metodología

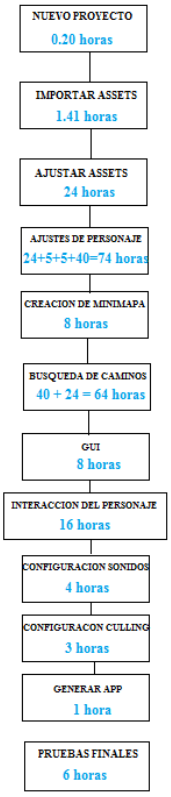


Figura 230. Tiempos de desarrollo empleados sin utilizar la metodología

CONCLUSIONES:

1. Se necesita realizar un estudio minucioso de las herramientas para la creación de Mundos Virtuales debido a que los motores 3D son compatibles con herramientas de modelado y animación siempre y cuando los modelos exportados por la herramienta de modelado tengan formatos compatibles con el motor 3D, o a su vez este realice cálculos internamente para transformarlo a un formato propio.
2. La guía metodológica propuesta en el presente estudio permite la correcta integración de un motor 3D con una herramienta de modelado y animación para la creación de Mundos Virtuales.
3. Es necesario verificar la metodología propuesta tanto en cada una de las fases así como una validación final del cumplimento general de la misma.
4. Tanto en la aplicación genérica como en el caso práctico del Campus Virtual de la ESPOCH, se puede ver que el usuario puede interactuar con el Mundo Virtual generado; esto se logró al aplicar la metodología creada en el estudio.
5. El Campus Virtual interactivo de la ESPOCH es accesible desde la Web; se puede acceder a él desde cualquiera de los navegadores existentes.
6. La aplicación generada funciona a una velocidad de 50 fps lo cual se encuentra dentro de los rangos de prueba que se obtuvieron de la aplicación genérica realizada.
7. El uso de un documento estándar de control basado en parámetros de ingeniería de software (Anexo 3) permite controlar el avance quincenal del proyecto y los cambios que en él se presentaron buscando así organizar de mejor manera el desarrollo del proyecto.

RECOMENDACIONES:

1. Al seleccionar la herramienta de modelado y el motor de video juegos para un proyecto, se debe tener en cuenta que los archivos que se exportan en la herramienta de modelado tengan un formato que sea aceptado por el motor.
2. La guía metodológica descrita puede ser aplicada en cualquier proyecto que se desee. Se deberá seguir paso a paso la presente metodología para obtener resultados óptimos.
3. Importar los modelos y texturas de la mejor calidad en el menor tamaño posible para que mejore la calidad del proyecto.
4. Para la búsqueda de caminos puede utilizarse cualquier algoritmo de búsqueda; se recomienda que sea usado el algoritmo que se propone en el presente estudio.
5. Para que la aplicación pueda ser publicada en la Web, es necesaria la utilización de un motor que lo permita. Revisar en la página del fabricante si el motor seleccionado lo permite antes de empezar la implementación de la metodología.
6. Ampliar los estudios realizados sobre las herramientas y los algoritmos de inteligencia artificial.

RESUMEN:

La presente investigación generó una metodología, para integrar herramientas de animación y modelado con motores 3D, optimizando la aplicación generada para publicarla en la Web. El caso práctico será la creación del Campus Virtual de la Escuela Superior Politécnica de Chimborazo.

Se utilizó un método de deductivo; partiendo de conceptos básicos relativos al tema, para continuar con una descripción de herramientas existentes en el mercado. Posteriormente se describió la metodología creada que es aplicable a cualquier herramienta; en este caso se escogieron como herramientas de prueba Autodesk Maya 2014 y Unity 3D que son flexibles y permiten la implementación de las aplicaciones generadas en la Web.

A continuación se creó una aplicación genérica y se realizó una comprobación de los parámetros verificables, como son cantidad y tamaño de texturas, número de luces y vértices, complejidad del material y ubicación de la cámara; de dicha comparación se obtuvo un rango de valores entre los cuales debe funcionar la aplicación final.

Se analizaron los datos obtenidos del Campus Virtual de la ESPOCH y los valores encontrados están dentro del rango de aceptabilidad, comprobándose que la metodología aplicada es correcta al 100% y puede ser aplicada a cualquier Mundo Virtual.

Aunque la metodología se probó en herramientas específicas, es válida para cualquier herramienta existente en el mercado; se recomienda que los programadores tomen en cuenta cual es el alcance del proyecto y la compatibilidad con los sistemas operativos para la selección de las mismas.

SUMMARY:

The present research generated a methodology to integrate 3D modeling and animation tools with game engines, which optimized the generated application to publish on the Web. The methodology was proved implementing a Virtual Campus of the Escuela Superior Politécnica de Chimborazo.

It was used a deductive method, starting with basic concepts related to the topic, continuing with a description of existing tools in the market. After that the created methodology was descripted in a way it can be applied to any tool, but Autodesk Maya 2014 and Unity3D were chosen as test tools because of their flexibility and allowing build applications to publish on Web.

Then a generic application was created; after that parameters such as number and size of textures, number of lights and vertices, complexity of the material and location of the camera where checked, in order to find a range of values to test any application.

Data obtained from ESPOCH´s Virtual Campus was analyzed and the values found are within the range of acceptability, which proves that the methodology used is functional at 100% and it could be applied to any virtual world.

It should be noted that even the methodology was tested in specific tools, it is valid for any existing tool on the market. The scope of the project must be considered besides the compatibility with operative systems in order to choose the correct tools.

GLOSARIO:

- **Asset Store:** Tienda de Unity en donde se pueden descargar assets gratuitamente o previo pago.

- **Avatar:** Representación de un usuario en un Mundo Virtual.

- **Bake:** Opción del motor Unity que hace referencia a la preparación de datos precalculados relativos a luces u optimización.

- **Caja de colisión:** Una caja de colisión (colisiones) o collider en inglés, es información que se le agrega a un asset dentro de una escena para que se realice el cálculo de colisiones y éste pueda chocar contra objeto.

- **Constructive Solid Geometry:** Técnica que se usa para el modelado de objetos de superficies complejas.

- **DirectX**: Es una API que ofrece herramientas para facilitar el desarrollo de tareas multimedia, como por ejemplo video juegos.

- **Fps (frames per second**): Unidad de medida que se usa para describir cuantos frames, o cuadros, se reproducen en un segundo. Los fps describirán la calidad de una aplicación multimedia.

- FPS (First Person Shooters): Tipo de video juegos en los que el usuario controla y ve el mundo como el personaje principal.

- **GNU/GPL:** Es una licencia de tipo gratuito y de libre distribución de aplicaciones software, que ofrece la libertad de usarlo, estudiarlo, copiarlo, modificarlo, entre otras.
- High-Order Surfaces: Método que permite la renderización en video juegos, especialmente en su terreno.

- **HMD:** Head-Mounted Display. Dispositivo en forma de casco o lentes que permite reproducir imágenes muy cerca de los ojos.

- **IDE:** Entorno de Desarrollo Integrado. Conjunto de herramientas que incluye un editor de código, compilador, depurador y constructor de aplicación.

- **Image based modeling (IBM):** Método con el cual se generan modelos 3D a partir de varias imágenes 2D.

- **Mallas poligonales:** Conocidos en inglés como mesh; es una superficie formada por polígonos con varios vértices, ángulos y caras (por lo menos 3) cada uno en su lugar correspondiente para formar un modelo en 3 dimensiones.

- **Metadata:** También conocidos como metadatos, se los conoce como “datos acerca de los datos”; son datos que permite encontrar datos que referencian objetos mediante índices.

- **MMORPG`s:** Juegos en los que múltiples jugadores pueden introducirse en un Mundo Virtual al mismo tiempo y pueden interactuar entre ellos, casi siempre por medio de un navegador Web.

- **Mono Develop:** Entorno de programación de Unity que permite la programación principalmente en C# y otros lenguajes .NET; mediante este IDE se pueden controlar y programar los objetos que se encuentren en una escena de Unity.

- **Motion Track:** Es una técnica que captura movimiento con diferentes aplicaciones y los transporta a computadoras para con esa información animar modelos 3D.

- **Normalmap:** Método mediante el cual se le pueden agregar detalles en relieve e iluminación a un objeto modelado en 3D.

- **NPC:** Hace referencia a un personaje no jugador, es decir todo aquel personaje que dentro de un juego es controlado por la inteligencia artificial del mismo en lugar de ser controlado por el usuario.

- **NURBS:** B-splines racionales no uniformes. Modelo matemático que permite representar superficies curvas en aplicaciones multimedia especialmente juegos de video.

- **OpenGL:** Es un API que cuenta con funciones y métodos, utilizada para la creación de aplicaciones que incluyen gráficos 2D y 3D.

- **Rigging:** Añadir huesos a un modelo para que posteriormente se le pueda agregar animación.

- **Shader**: Es un procedimiento de sombreado e iluminación mediante el cual se agregaa renderizado a cada uno de los vértices.

- **Skinning:** Proceso en el cual se le agrega relaciones entre los huesos creados en el Rigging y los vértices.

- **Splash screen**: Imagen que se muestra mientras un video juego se carga.

- **Uv:** Son coordenadas que se asignan a cada vértice de un modelo que determinarán como se proyectará una textura sobre un modelo.

ANEXOS

ANEXO 1

ENCUESTA

MUNDOS VIRTUALES

*Obligatorio

EDAD *

SEXO *

LUGAR DE UBICACION *

Pais

LUGAR DE UBICACION *

Ciudad

CONOCE LA ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO (ESPOCH) *

- Si
- No

PUEDE UBICAR LOS EDIFICIOS PRINCIPALES

Como Edificio Central, Biblioteca, Auditorio, etc

- Si
- No

PUEDE UBICAR LAS FACULTADES Y ESCUELAS DE LA ESPOCH

- Si
- No

CONSIDERA QUE SI EXISTIESE UN CAMPUS VIRTUAL DE LA ESPOCH LE SERIA UTIL PARA CONOCERLA

- Si

- No

¿POR QUE?

- Ahorro de tiempo
- Ahorro de dinero
- No viajar para conocerla
- Conocerla Inmediatamente
- Conocerla previamente
- Otro:

HA VISITADO DE MANERA VIRTUAL ALGUNA UNIVERSIDAD DENTRO DE ECUADOR

- Si
- No

HA VISITADO DE MANERA VIRTUAL ALGUNA UNIVERSIDAD EN EL EXTERIOR

- Si
- No

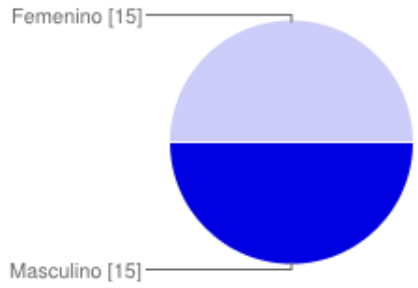
Enviar

RESULTADOS

EDAD

17 22 23 24 25 20 18 15 16 21

SEXO



Masculino	15	50%
Femenino	15	50%

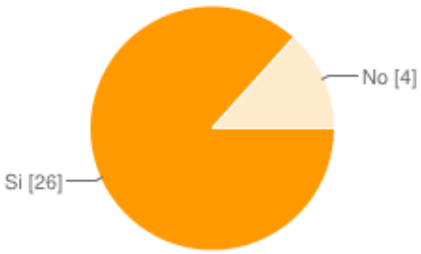
LUGAR DE UBICACION (País)

Ecuador Nicaragua

LUGAR DE UBICACION (Ciudad)

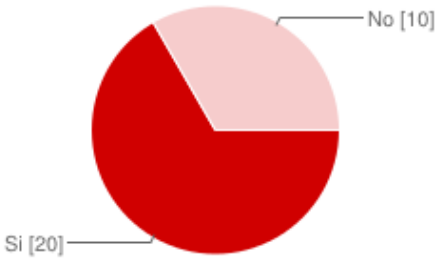
Guaranda Guayaquill loja Quito ambato Riobamba santo domingo Carchi Guano Coca Tena Banios Lago Agrio Masaya Esmeraldas

CONOCE LA ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO (ESPOCH)



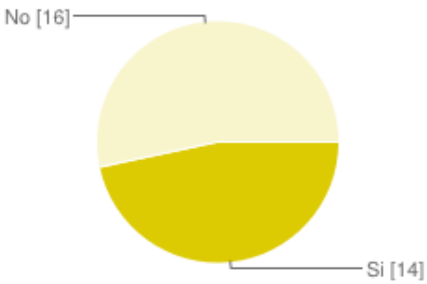
Si	**26**	87%
No	**4**	13%

PUEDE UBICAR LOS EDIFICIOS PRINCIPALES



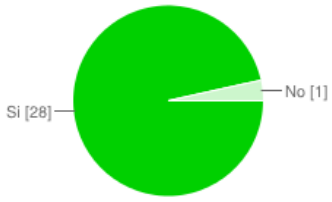
Si	**20**	67%
No	**10**	33%

PUEDE UBICAR LAS FACULTADES Y ESCUELAS DE LA ESPOCH



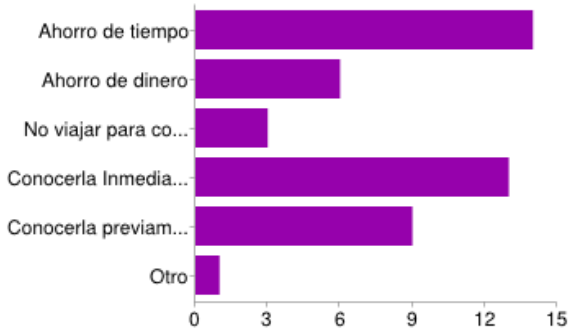
Si	**14**	47%
No	**16**	53%

CONSIDERA QUE SI EXISTIESE UN CAMPUS VIRTUAL DE LA ESPOCH LE SERIA UTIL PARA CONOCERLA



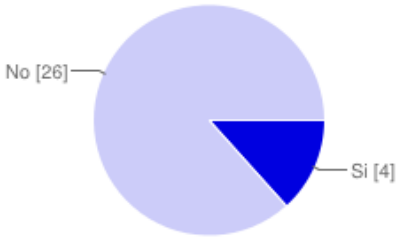
Si	28	97%
No	1	3%

¿POR QUE?



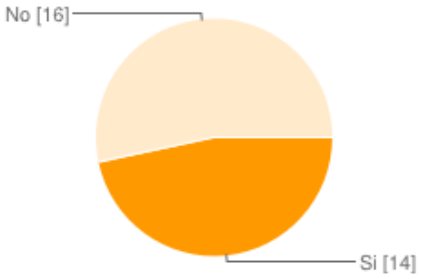
Ahorro de tiempo	14	30%
Ahorro de dinero	6	13%
No viajar para conocerla	3	7%
Conocerla Inmediatamente	13	28%
Conocerla previamente	9	20%
Otro	1	2%

HA VISITADO DE MANERA VIRTUAL ALGUNA UNIVERSIDAD DENTRO DE ECUADOR



Si	4	13%
No	26	87%

HA VISITADO DE MANERA VIRTUAL ALGUNA UNIVERSIDAD EN EL EXTERIOR



Si	14	47%
No	16	53%

ANEXO 2

Algoritmo A*

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

public class AStar{

    List<NodoEspacial> openSet = new List<NodoEspacial>();
    List<NodoEspacial> closedSet = new List<NodoEspacial>();
    public List<NodoEspacial> path;

    private NodoEspacial GetLowestF()
    {
        int index = 0;
        float val = openSet[0].F;
        for (int i = 1; i < openSet.Count; i++)
        {
            if (openSet[i].F < val)
            {
                index = i;
                val = openSet[i].F;
            }
        }

        NodoEspacial node = openSet[index];
        openSet.RemoveAt(index);
        closedSet.Add(node);

        return node;
    }

    private bool InClosed(NodoEspacial node)
    {
        for (int i = 0; i < closedSet.Count; i++)
        {
            if (node == closedSet[i])
            {
                return true;
            }
        }
        return false;
    }

    private bool InOpen(NodoEspacial node)
    {
        for (int i = 0; i < openSet.Count; i++)
        {
            if (node == openSet[i])
            {
                return true;
            }
        }
```

```
        return false;
    }

public int hasRelation(NodoEspacial start,NodoEspacial goal)
{
int index = -1;

for(int i=0;i<start.relaciones.Length;i++)
{
if(start.relaciones[i]==goal)
return i;
}
return index;
}
    public void Start(NodoEspacial start,NodoEspacial goal)
    {
        openSet = new List<NodoEspacial>();
    closedSet = new List<NodoEspacial>();
path = new List<NodoEspacial> ();
start.G = 0;
int index = hasRelation(start,goal);
if(index>-1)
{
start.H = start.distancias[index];
}
else
{
        start.H = Vector3.Distance(start.transform.position, goal.transform.position);
}
        start.F = start.G + start.H;

        openSet.Add(start);

        float tentativeGScore;
        bool tentativeIsBetter;
        path = new List<NodoEspacial>();
        while (openSet.Count > 0)
        {
            NodoEspacial x = GetLowestF();

            if (x == goal)
            {
                NodoEspacial node = x;
                while (node != null)
                {
                    path.Add(node);
                    node = node.parent;
                }
                //reconstruir path
            }

            for (int i = 0; i < x.relaciones.Length; i++)
            {
                NodoEspacial y = x.relaciones[i];
                if (InClosed(y)) continue;
                tentativeGScore = x.G + x.distancias[i];//Vector3.Distance(x.transform.position,
y.transform.position);
                if (!InOpen(y))
```

```
            {
                openSet.Add(y);
                tentativeIsBetter = true;
            }
            else
            {
                if (tentativeGScore < y.G)
                {
                    tentativeIsBetter = true;
                }
                else
                {
                    tentativeIsBetter = false;
                }
            }

            if (tentativeIsBetter)
            {
                y.G = tentativeGScore;
index = hasRelation(y,goal);
if(index>-1)
{
                y.H = y.distancias[index];
}
else
{
y.H = Vector3.Distance(y.transform.position,goal.transform.position);
}
                y.F = y.G+y.H;
                y.parent = x;
            }
        }
    }
  }
                                                    }
```

ANEXO 3

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FORMATO DE VERIFICACIÓN Y VALIDACIÓN DE SOFTWARE

CODIGO: GCS-01-2014

TIPO DE CONTROL: **GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE**

NOMBRE DE LA APLICACIÓN: ……

RESPONSABLE: ……

DIRIGIDO A/USUARIO FINAL: ……

FECHA DE REALIZACIÓN: INICIO: …….. FIN: ……..

PORCENTAJE DE USO DE LA METODOLOGIA: …… (%)

Nr.	Actividades generales	Productos	Tiempo h/hombre	Observación o Anexo/Refer.

OBSERVACIONES GENERALES:

Riobamba, dd,mm,aaaa

---------- ---------- ----------

(f)FIRMA TECNICO (f) FIRMA REVISOR (f) FIRMA DIRECTOR

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FORMATO DE VERIFICACIÓN Y VALIDACIÓN DE SOFTWARE

CODIGO: V&V- 01-2014

TIPO DE CONTROL: **VERIFICACIÓN Y VALIDACIÓN DE FASE DEL SISTEMA**

NOMBRE DE LA FASE: ……

RESPONSABLE: ……

DIRIGIDO A/USUARIO FINAL: ……

FECHA DE REALIZACIÓN: INICIO: …….. FIN: ……..

PORCENTAJE DE USO DE LA METODOLOGIA: …… (%)

Nr.	Tareas especificas	Subproductos	Tiempo h/hombre	Observación o Anexo/Refer.

OBSERVACIONES GENERALES:

Riobamba, dd,mm,aaaa

---------- ---------- ----------

(f)FIRMA TECNICO (f) FIRMA REVISOR (f) FIRMA DIRECTOR2.-

BIBLIOGRAFÍA

[1] MANETTA, C. Y BLADE , R., Glosarry of virtual reality terminology., 1ª. ed., Denver, CO - USA., University of Colorado., 1995., p.p. 35-39

[2] ¿QUÉ ES INTERACTIVIDAD?
http://blogs.enap.unam.mx/asignatura/francisco_alarcon/wp-content/uploads/2011/06/interactividad.pdf.
2013 - 9- 4

[3] MOTOR DE VIDEOJUEGOS
http://www.ecured.cu/index.php/Motor_de_videojuego
2013 - 9 - 4

[4] GAMES SOUND
http://www.gamessound.com/learning.htm
2013 - 9 - 5

[5] RABIN, **S.,** *AI Game Programming WISDOM.*, 1ª. ed., Rockland, MA - USA., Charles River Media, Inc., 2002., pp. 105-107

[6] REALIDAD VIRTUAL
http://www.jeuazarru.com/docs/RealidadVirtual.pdf.
2013 - 9 - 4

[7] BROOKS, F., What's Real About Virtual Reality?, *IEEE Computer Graphics and Applications.,* 1ª. ed., Los Alamitos, CA - USA., IEEE Computer Society Press., pp. 16-27, 1999.

[8] GRÁFICOS 3D POR COMPUTADORA
http://www.ecured.cu/index.php/Gr%C3%A1ficos_3D_por_computadora
2014 - 2 - 18

[9] VIRTUAL WORLD
http://virtualworldblog.com/?p=102.
2013 - 10 - 5

[10] WHAT IS A GAME ENGINE?
http://www.gamecareerguide.com/features/529/what_is_a_game_.php?.
2013 - 10 - 9

[11] C. DELRIEUX Y J. GAMBINI, Modelos de Iluminación y Sombreado, de *COMPUTACION GRAFICA.*, 1ª. ed., Bahía Blanca - Argentina., JEITICS., 2003., pp. 197-230

[12] Graficación por computadora
https://sites.google.com/site/grafcomputacional/animacion-3d/iluminacion.
2013 - 9 - 24

[13] Bump Mapping
http://freespace.virgin.net/hugo.elias/graphics/x_polybm.htm.
2013 - 12 -3

[14] UNITY 3D

http://spanish.unity3d.com/.

2014 - 1- 17

[15] UNREAL ENGINE

http://www.unrealengine.com/.

2014 - 1 - 18

[16] 3DS MAX

http://www.autodesk.es/products/autodesk-3ds max/overview.

2014 - 1- 16.

[17] MAYA

http://www.autodesk.es/products/autodesk-maya/overview.

2014 - 1 - 16

[18] BLENDER

http://www.blender.org/.

2014 - 1 -15

[19] CINEMA 4D STUDIO

http://www.maxon.net/es/products/cinema-4d-studio.html.

2014 - 1 -16

[20] CRYENGINE

http://mycryengine.com/.

2014 -1 - 21