UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

CAMPUS D'ALCOI

Universitat Politècnica de València

Campus d'Alcoi

Desarrollo de videojuego 3D con Unity

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: José Luis Morant Capellino

Tutor: Jordi Joan Linares Pellicer

Curso 2016-2017

Resumen

El proyecto consiste en el desarrollo de un videojuego en 3D para computador simulando la atracción gravitatoria de un planetoide donde se desarrollará la acción. Para cumplir el objetivo se utilizarán herramientas de tratamiento de imagen para texturas, modelado 3D para objetos y personajes y editores de efectos de sonido y música. Como motor del videojuego multiplataforma se hará uso de Unity 3D.

Palabras clave: Unity 3D, videojuego, modelado 3D, texturizado, sonido.

Abstract

The project focuses on the development of a 3D computer video game which simulates the gravitational attraction of a planetoid where the action will be carried out. To achieve this target, it will be used image processing tools for textures, 3D modeling for objects and characters and editors of sound effects and music. As cross-platform game engine it will be used Unity 3D.

Keywords : Unity 3D, video game, 3D modeling, textured, sound.

Índice de contenidos

1. Introducción 13
   1.1. Motivación 13
   1.2. Descripción del proyecto 13
   1.3. Mercado actual de los videojuegos para computador 13
   1.4. Objetivo principal del proyecto 15
2. Herramientas de trabajo 16
   2.1. 3ds Max 16
   2.2. Mudbox 17
   2.3. Photoshop CC 18
   2.4. Illustrator CC 19
   2.5. MAGIX Music Maker 20
   2.6. Leshy SFMaker 21
   2.7. Unity 3D 22
   2.8. Visual Studio Community 2017 24
3. Planificación del proyecto 25
4. Desarrollo del videojuego 27
   4.1. Problema principal 27
   4.2. Crear un planetoide 28
   4.3. Sistema de Waypoints 29
   4.4. Modelado de un enemigo 30
      4.4.1. Base del modelo 30
      4.4.2. Modelado a alto nivel 31
      4.4.3. Generar mapa de normales 32
      4.4.4. Reducción de polígonos 33
      4.4.5. Modelo a bajo nivel con mapa de normales 34
      4.4.6. Animación del enemigo 37
5. Texturizado 38
   5.1. Texturizando una esfera 38
6. Sistema de párticulas 42
   6.1. Definición 42
   6.2. Nubes 42
   6.3. Partículas emanando de la luz 44

7. Programación 45
7.1. Rotación del planetoide 45
7.2. Rellenar una esfera de objetos 47
7.3. Intensidad de luz variable 48
7.4. Sistema de waypoints 50
7.5. Cámara centrada al planetoide 51
7.6. Planetoide y objetos atraídos 52
8. Música 53
9. Trabajo futuro 54
10. Conclusiones 55
11. Bibliografía 56

Índice de figuras

Figura 1 - Entorno de trabajo de 3ds Max. 16
Figura 2 - Entorno de trabajo de Mudbox. 17
Figura 3 - Entorno de trabajo de Photoshop CC. 18
Figura 4 - Entorno de trabajo de Illustrator. 19
Figura 5 - Entorno de trabajo de MAGIX Music Maker. 20
Figura 6 - Previsualización de Leshy SFMaker (https://www.leshylabs.com/apps/sfMaker) 21
Figura 7 - Entorno de trabajo de Unity 3D. 22
Figura 8 - Entorno de trabajo de Visual Studio Community 2017 24
Figura 9 – La tierra no es plana. Nuestro planetoide tampoco. 27
Figura 10 - Las personas no deben caminar con su cabeza. 28
Figura 11 - Prueba técnica sobre la atracción de un planetoide. 28
Figura 12 - Sistema de Waypoints 29
Figura 13 - Objetos haciendo de Waypoints sobre el camino. 29
Figura 14 - Base poligonal de un enemigo con la que trabajar. 30
Figura 15 - Modelado en alto detalle 31
Figura 16 - Mapa de normales de un enemigo. 32
Figura 17 - Modelo a bajo nivel. 33
Figura 18 - Modelo a bajo nivel en Unity 3D. 34
Figura 19 - Modelo a bajo nivel en Unity 3D con mapa de normales aplicado. 35
Figura 20 - Previsualización del enemigo en Unity 3D. Planetoide modelado con Mudbox. 36
Figura 21 - Aplicando un esqueleto bípedo al modelo. 37
Figura 22 - Textura realizada con photoshop. 38
Figura 23 - Textura aplicada sobre una esfera. 39
Figura 24 - Textura con el filtro de coordenadas polares aplicado. 40
Figura 25 - Textura con el filtro de coordenadas polares aplicado a una esfera. 40
Figura 26 - Mapamundi de la Tierra. Fuente: es.wikipedia.org. 41
Figura 27 - Atmosfera del planetoide. 42
Figura 28 - Sistema planetario, compuesto por el planetoide, la estrella y un satélite. 43
Figura 29 - Nuestra estrella ocultándose detrás de las nubes. Parte oscura del planetoide. 43
Figura 30 - Satélite ocultándose entre las nubes. Parte iluminada del planetoide. 44
Figura 31 - Partículas emanando de la luz. 44
Figura 32 - Esfera al rededor del planetoide. 45
Figura 33 - Código que nos permite rotar la esfera que controla la cámara. 46
Figura 34 – Código que nos permite añadir objetos aleatoriamente sobre una esfera. 47
Figura 35 - Cambio de intensidad de la iluminación. 48
Figura 36 - Código que nos permite cambiar la intensidad de la luz. 49
Figura 37 - Código que nos permite seguir un recorrido de waypoints. 50
Figura 38 - Código que nos permite centrar la cámara en el planetoide. 51
Figura 39 - Código que nos permite atraer a otros objetos a nuestra posición. 52
Figura 40 - Código que nos permite asociar el objeto para ser atraído al planetoide. 52

Figura 41 - Representación gráfica de las pistas al inicio de la música del primer nivel. 53

Índice de gráficos

Gráfico 1 - Representación gráfica del nivel de ventas de Steam por mes. 14
Gráfico 2 - Diagrama de Gantt del proyecto... 26

Índice de tablas

Tabla 1 - Representación del nivel de ventas de Steam por mes en formato tabla. 14
Tabla 2 - Modalidades de licencia de Unity 3D. 23
Tabla 3 - Planificación del proyecto por fecha de inicio, fin y días. 25

1. Introducción

1.1. Motivación

Tras haber cursado la asignatura de Introducción a la Programación de Videojuegos que imparte Jordi Linares y ser un consumidor y jugador de videojuegos desde que tengo uso de razón, este es un mundo que siempre me ha fascinado y quería poder ahondar más a fondo en el desarrollo de los videojuegos para poder comprender todos los puntos de los que se constituyen y contemplar la posibilidad de poder dedicarme profesionalmente al sector en el futuro.

1.2. Descripción del proyecto

El proyecto consiste en el desarrollo de un videojuego utilizando como motor Unity 3D.

Este videojuego pertenecerá al subgénero de estrategia en tiempo real conocido como Tower Defense pero cambiando el clásico entorno plano que los caracteriza por uno relativamente esférico, concretamente un planetoide. Haciendo uso de fuerzas de gravedad que mantendrán a los personajes y objetos atraídos al cuerpo celeste.

1.3. Mercado actual de los videojuegos para computador

Actualmente, aunque compres un videojuego físicamente lo más probable es que tengas que pasar igualmente por una plataforma de distribución digital como Steam, Origin si el juego es de EA (FIFA, Battlefield, …) o battle.net para los juegos de Blizzard (Starcraft II, Diablo III, World of Warcraft, …), Uplay… Aunque existen otros menos populares como GOG.com que ofrecen sus videojuegos sin DRM (gestión de derechos digitales).

Yo voy a centrarme en las estadísticas de Steam durante el pasado año (2016) ya que es la plataforma más popular, que más cuota de mercado posee y en el que me gustaría poder acabar subiendo mi contenido

El origen de los datos proviene de steamspy.com ya que Steam no facilita datos estadísticos que no sean del tipo: los videojuegos más jugados, cantidad de jugadores online, plataformas más usadas, hardware empleado, … Por lo que los datos son prácticamente 100% reales, pero no se pueden catalogar como oficiales.

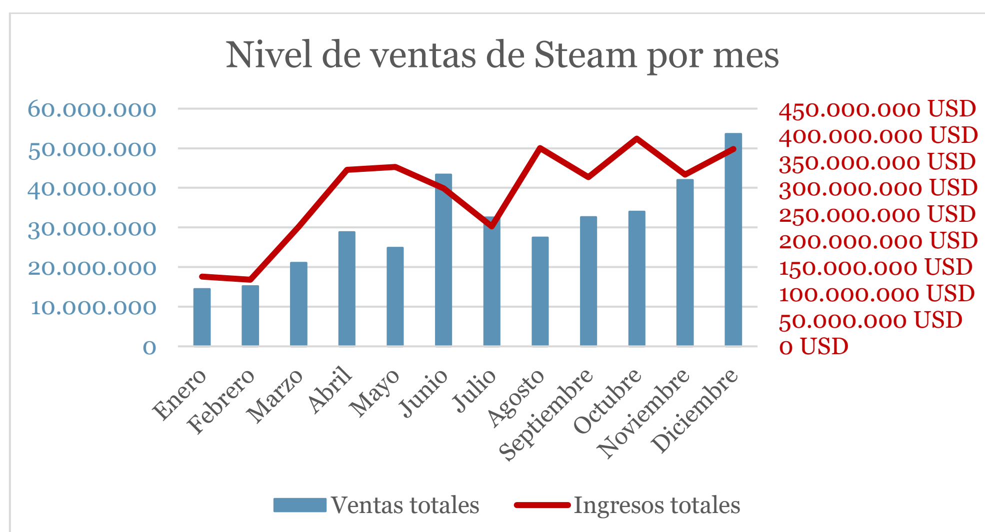

Gráfico 1 - Representación gráfica del nivel de ventas de Steam por mes.

Los aumentos sustanciales del nivel de ventas en junio y diciembre se deben a que Steam ofrece ofertas (Summer Sale y Winter Sale) en esas fechas por eso los ingresos esos meses son equiparables a los otros meses aun incrementando las ventas.

Mes	Ventas totales	Ingresos totales
Enero	14.454.157	132.221.415 USD
Febrero	15.216.088	126.161.337 USD
Marzo	21.042.875	226.422.654 USD
Abril	28.786.282	334.240.215 USD
Mayo	24.872.667	339.628.195 USD
Junio	43.362.476	299.276.763 USD
Julio	32.516.107	226.916.339 USD
Agosto	27.458.741	375.568.452 USD
Septiembre	32.612.019	320.002.608 USD
Octubre	33.998.270	393.246.237 USD
Noviembre	41.944.088	324.901.740 USD
Diciembre	53.622.778	373.562.956 USD
TOTAL	**369.886.548**	**3.472.148.911 USD**

Tabla 1 - Representación del nivel de ventas de Steam por mes en formato tabla.

Steam al finalizar el año 2016 vendió un total de 369.886.548 videojuegos y recaudó 3.472.148.911 $.

Durante este año se triplico el número de ventas de videojuegos en modo “Early Acces” respecto del año anterior 2015, produciéndose mucha polémica al respecto ya que está modalidad permite comprar y jugar al videojuego sin que esté acabado y muchas desarrolladoras dejaban su producto inacabado al llegar al nivel de ventas deseado. Esto ha llevado a la gente a ser más prudente ante este tipo de modalidades.

1.4. Objetivo principal del proyecto

El objetivo principal del proyecto será el de adquirir y/o mejorar conocimientos sobre todos los puntos importantes de los que consta un videojuego: Diseño de objetos, personajes y niveles, sonido, programación y aspecto visual. Utilizando las herramientas oportunas para cada campo y como motor para videojuegos Unity 3D.

2. Herramientas de trabajo

2.1. 3ds Max

Es un software propietario desarrollado en el lenguaje de programación C Sharp por Autodesk para la creación de gráficos y animación 3D y compilado para ser usado en el sistema operativo Microsoft Windows.

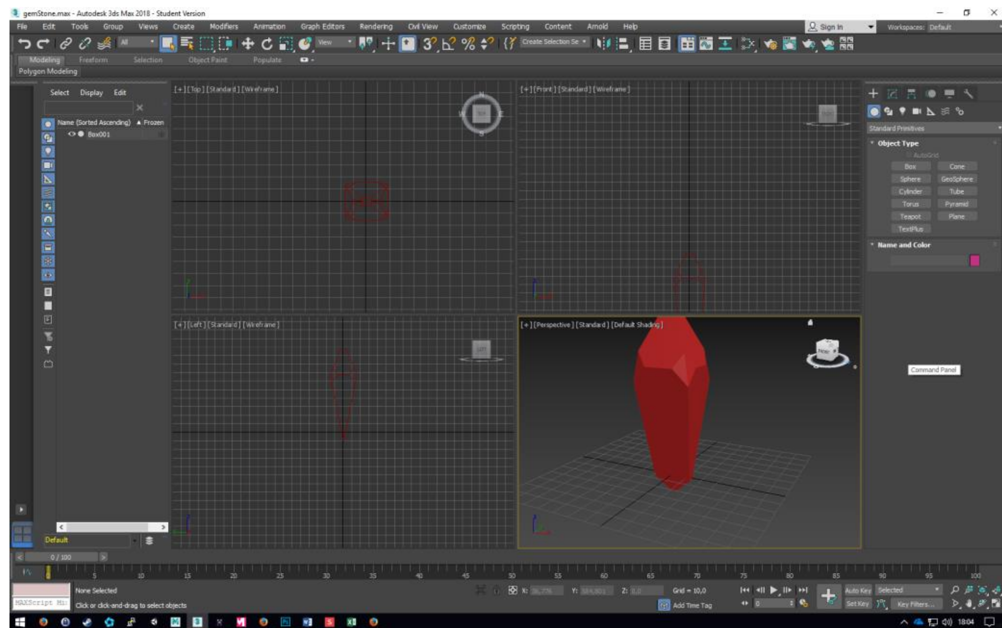

Figura 1 - Entorno de trabajo de 3ds Max.

Me he decidido por 3ds Max para la realización de objetos de escenario que no sean en principio muy redondeados, o la base de estos con pocos detalles para después exportarlos y mejorarlos. Así, como el encargado de realizar los huesos (bones), esqueletos y animaciones de los personajes.

Yo uso la 'Student Version' (Versión para estudiantes) que es gratuita y respeta todas las opciones de la versión original siempre y cuando seas estudiante.

En caso de no ser estudiante hay varias modalidades de pago en forma de suscripción. 200€ en la opción mensual o 1.600€ por la anual.

Existen alternativas gratuitas como Blender, pero me he decantado por esta al ser la más profesional y completa para el desarrollo de videojuegos.

2.2. Mudbox

Mudbox al igual que 3ds Max está desarrollado por Autodesk y es un software de modelado 3d, texturado y pintura digital. Es un software propietario programado en C++ y es compatible tanto en Sistemas Microsoft Windows, Mac OS X como Linux.

Figura 2 - Entorno de trabajo de Mudbox.

La principal diferencia de Mudbox respecto 3ds Max es que su uso principal es para crear objetos con mucho detalle y cantidad de polígonos, así como su forma de trabajar que es similar al del modelado tradicional sobre barro o arcilla.

Me he decantado por el para el desarrollo de personajes y objetos que requieran de mucho detalle como pueden ser los propios planetoides de los que constará el videojuego.

En este caso, el software más conocido para este fin es ZBrush de Pixologic y es con el que realmente empecé a probar e incluso llegué a desarrollar un personaje completo, pero tuve problemas al trabajar en conjunto de 3ds Max.

La diferencia entre los dos no la he notado en cuanto a resultados, pero la interface entre los dos si que es totalmente diferente.

La licencia de Zbrush cuesta 740€, mientras que Mudbox al igual que 3ds Max es mediante suscripción, con un precio de 12,10€ mensuales o 102,85€ de forma anual.

En mi caso hago uso de la versión gratuita para estudiantes como con 3ds Max.

2.3. Photoshop CC

Es un editor de gráficos rasterizados y retoque fotográfico propietario mundialmente conocido. Su desarrollador es Adobe Systems Incorporated y está programado en el lenguaje de programación C++ y es compatible tanto en sistemas Microsoft Windows como en los sistemas Apple Macintosh.

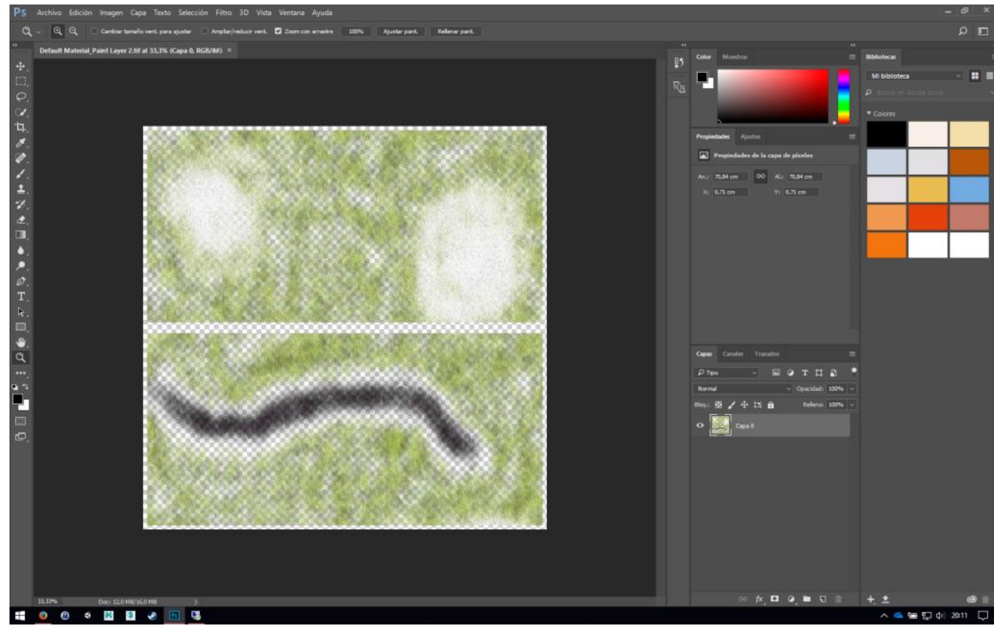

Figura 3 - Entorno de trabajo de Photoshop CC.

En este caso, Photoshop lo usaré para crear las texturas de los modelos 3D, imágenes de partículas, imágenes de decoración de estado e interfaz gráfica del videojuego.

Si hay alguna posibilidad de que se le acerque algún programa a Photoshop sería GIMP de software libre, pero la verdad que en este caso la diferencia entre los dos es abismal en cuanto a la cantidad de opciones y filtros, por lo que en este caso no he tenido ninguna duda en la elección. No quiero decir que GIMP sea un mal programa, sino que, si entras en profundidad con Photoshop, GIMP se te queda corto.

La licencia de uso de Photoshop consta de una suscripción mensual de 12,09€ e incluye Lightroom, para que se incluya todo el software perteneciente a Creative Cloud, hay dos modalidades: la de estudiantes o profesores por 19,66€ al mes o 24,19€ al mes si no perteneces a ninguno de estos colectivos.

Existe la posibilidad de bajarse Photoshop en periodo de prueba, pero este es solo de 7 días.

En cuanto a GIMP es software libre perteneciente a la fundación GNOME y sus autores son Peter Mattis y Spencer Kimball.

2.4. Illustrator CC

Es un editor propietario de gráficos vectoriales desarrollado al igual que Photoshop por Adobe Systems Incorporated. Está programado en el lenguaje de programación C++ y en principio fue desarrollado exclusivamente para sistemas Mac OS X, aunque desde hace tiempo está disponible para sistemas Microsoft Windows.

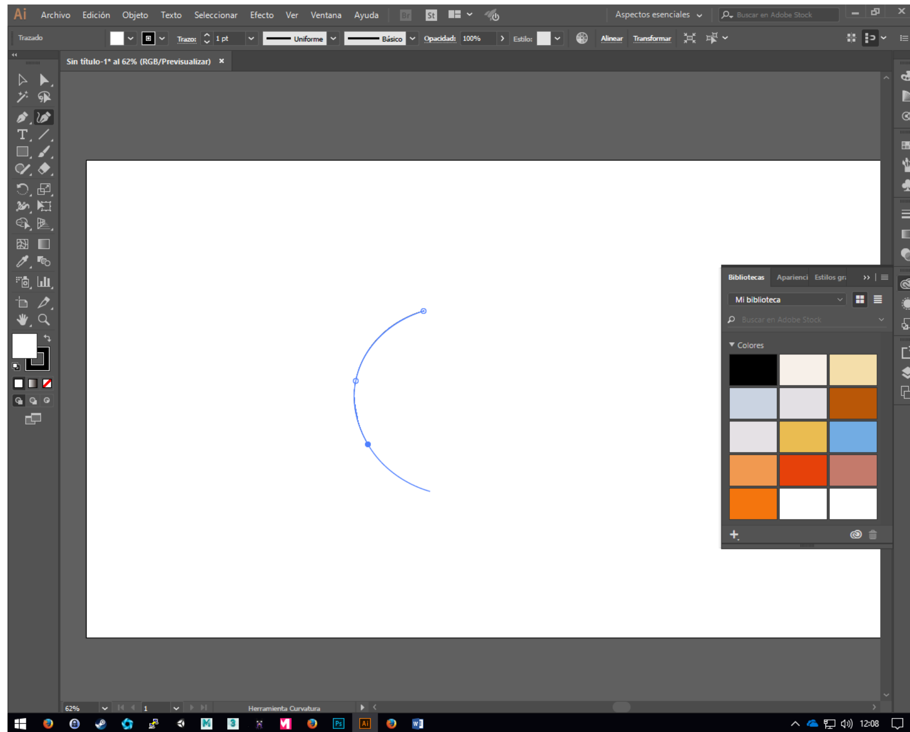

Figura 4 - Entorno de trabajo de Illustrator.

Illustrator me va a permitir hacer los dibujos de artísticos del videojuego, ya sea la portada, menús, o dibujo de personajes.

Si que es cierto que en este caso me podía servir la alternativa gratuita Inkscape, ya que en principio no voy a necesitar hacer trabajos impresos que es donde radica la principal diferencia entre los dos, puesto que Illustrator posee la licencia sobre los colores impresos de Pantone y más filtros CMYK.

Lo he escogido porque en el ámbito profesional es el que más se utiliza y puestos a ampliar mis conocimientos he preferido hacerlo sobre esta aplicación.

Hace tiempo usaba FreeHand que pertenecía a Macromedia y era líder indiscutible del sector de gráficos vectoriales, pero Adobe lo compró, e Illustrator junto con InDesign que está desarrollado más para maquetar, vendrían a ser la evolución de FreeHand.

En cuanto a la licencia se podría hacer uso de Creative Cloud como con Photoshop y tener todo el software de Adobe por 19,66€ al mes si perteneces al colectivo de estudiantes o profesores o 24,19€ al mes si no perteneces.

Ocurre el mismo caso que con Photoshop y el periodo de prueba consta de 7 días.

2.5. MAGIX Music Maker

Es un editor de audio, multipista desarrollado por Magix Software GmbH. Es de licencia propietaria y exclusivo para sistemas Microsoft Windows.

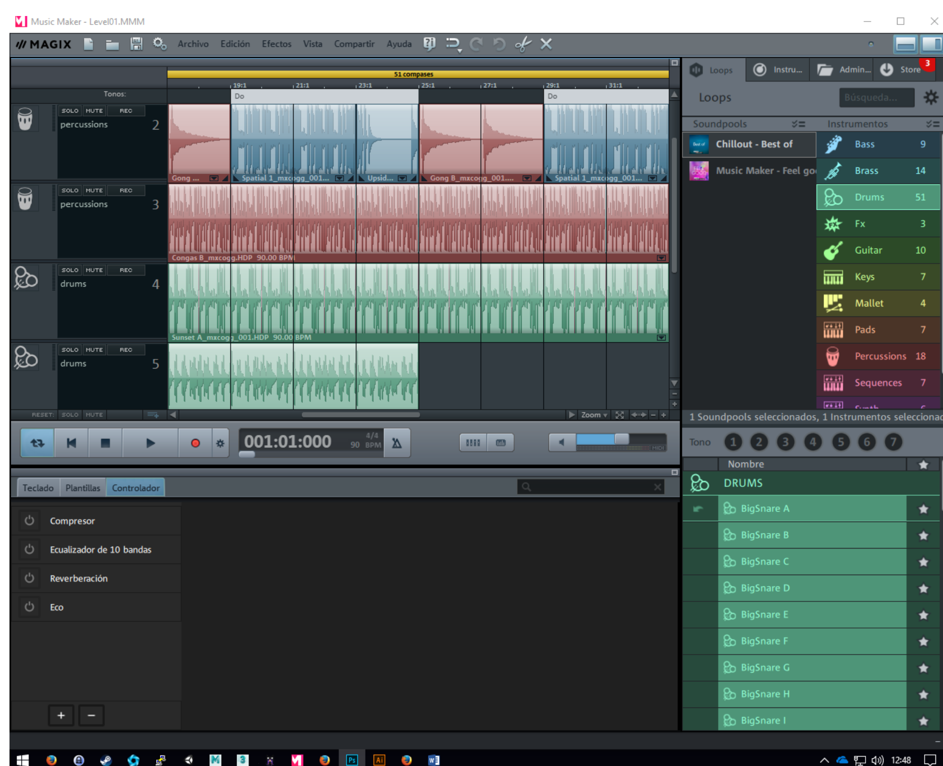

Figura 5 - Entorno de trabajo de MAGIX Music Maker.

He escogido MAGIX Music Maker para realizar la música de fondo tanto de niveles como de menú principal para el videojuego.

Si me he decantado por él, es porque ya poseía una licencia que incluía un bono para poder comprar más samples de audio por el valor de 30€. Tras ver comparativas con otros programas del sector, los veía bastante parecidos y la verdad es que en este caso parto básicamente sin conocimientos previos.

En principio el software se puede bajar de forma totalmente gratuita, pero viene con un set de samples de audio limitado. La forma de negocio que tiene es un apartado de la aplicación denominada Store (Tienda) donde puedes comprar Soundpools que vienen a ser lotes de audio de diferentes instrumentos y normalmente catalogados por estilos musicales.

2.6. Leshy SFMaker

Es un editor online que permite crear efectos de sonido. Está desarrollado por Leshy Labs LLC y tiene soporte para los navegadores Google Chrome y Mozilla Firefox, dejando fuera a Internet Explorer.

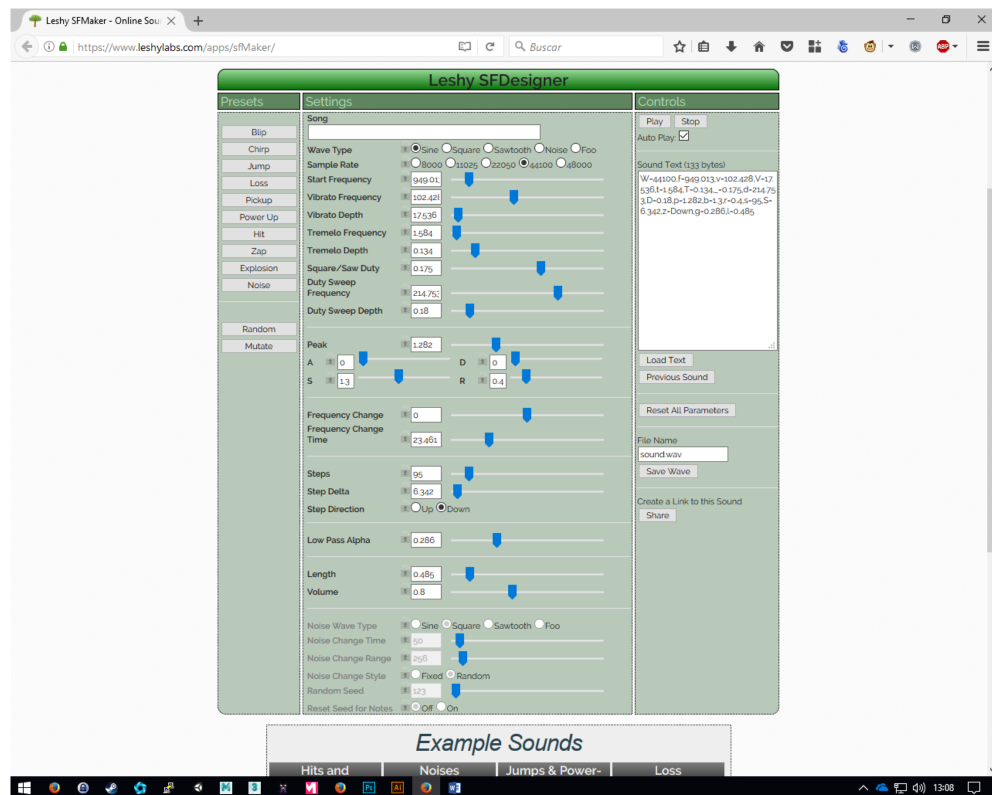

Figura 6 - Previsualización de Leshy SFMaker (https://www.leshylabs.com/apps/sfMaker)

El uso que le voy a dar es el de crear efectos de sonido tales como disparos, fuego, clic de botones, …

Lo empecé a utilizar cuando desarrollaba las prácticas de Introducción a la Programación de Videojuegos y me gustaron los resultados, así que voy a seguir utilizando para la realización del proyecto.

Es totalmente gratuito de utilizar y permite exportar los efectos de sonido realizados en formato Wave (.wav).

2.7. Unity 3D

Es un motor de videojuegos multiplataforma desarrollado por Unity Technologies. Está programado en C, C++ y C Sharp y es compatible en sistemas tanto Microsoft Windows, Mac OS X como Linux.

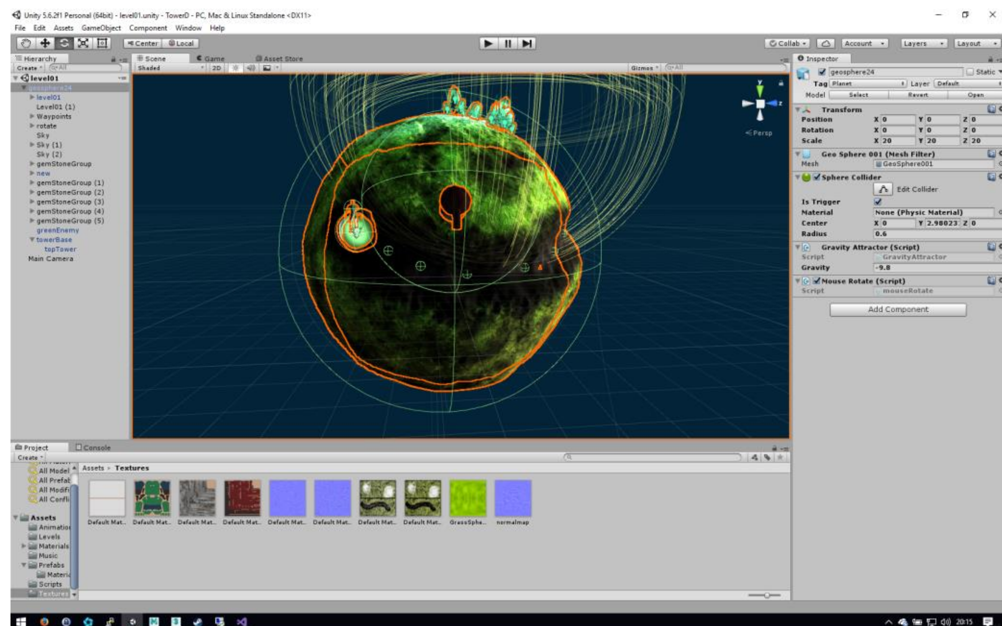

Figura 7 - Entorno de trabajo de Unity 3D

Unity 3D se va a encargar de englobar y darle sentido a todos los archivos creados previamente con la ayuda del software descrito en los puntos anteriores. Estos archivos conocidos en Unity 3D como 'Assets' se importarán en subdirectorios organizados a nuestro gusto bajo el directorio principal del mismo nombre.

La elección de Unity 3D como motor del videojuego es porque cumple los requisitos para el desarrollo del mismo y vengo de la experiencia de utilizarlo en la asignatura de Introducción a la programación de videojuegos.

Junto Unity 3D existen otros motores de videojuego en el mercado capaces de compilar para computador que rivalizan directamente como son: Unreal Engine y Cry Engine. Hay otros, pero en temas de 3D estos serían de forma accesible al público los más importantes.

La ventaja que posee Unity 3D es la basta documentación que posee y la principal pega que le pongo es que nativamente no soporta la edición de la maya de los objetos, teniendo que acudir a reeditar el modelado 3D del objeto mediante software externo, como pueda ser 3ds Max, aunque sea una modificación puntual e insignificante. Si que es cierto que en la 'Store' (Tienda) hay añadidos para poder incorporar esta característica, pero o tienes que pasar por caja o sino resulta limitado.

Unreal Engine también tiene un buen soporte en cuanto a comunidad y documentación, aunque menor que Unity 3D, y los acabados resultan ser más

fotorrealistas. Desde mi punto de vista sería muy buena elección para el desarrollo de un videojuego de tipo ‘shooter’.

CryEngine es muy potente y también ofrece características similares a Unreal Engine, pero es el más perjudicado en cuanto a documentación y apoyo de la comunidad.

Unity 3D cuenta con tres opciones diferentes de licencia: Personal, Plus y Pro.

Modalidad	Precio	Descripción
Personal	gratuito	• Para principiantes o estudiantes. • Todas las prestaciones básicas del motor. • No permite quitar el logo de Unity 3D al cargar el videojuego.
Plus	35$ / mes	• Essentials Pack • Pantalla de inicio personalizable • Informes de ejecución • Unity Analytics ampliado • Gestión flexible de puestos • Editor UI Skin de la versión Pro • Priority Cloud Builds
Pro	125$ / mes	• Essentials Pack • Todas las prestaciones de la versión Plus • Servicios de nivel Pro • Tienes a tu disposición planes con soporte Premium y acceso al código fuente • No se aplican restricciones a los ingresos ni a la recaudación de fondos

Tabla 2 - Modalidades de licencia de Unity 3D.

Unreal Engine es totalmente gratuito, pero se deberá pagar el 5% de las ganancias que obtengas una vez acumulados 3000$. En cuanto a CryEngine se puede pagar lo que veas justo por utilizarlo, como si no quieres pagar nada.

2.8. Visual Studio Community 2017

Es un IDE (Entorno de desarrollo integrado) programado en C++ y C Sharp y desarrollado por Microsoft. Posee las mismas características que la versión 'Professional' pero pensado para empresas de pequeño tamaño, estudiantes y desarrolladores de software libre. Puede ser instalado en Sistemas Microsoft Windows y MacOS.

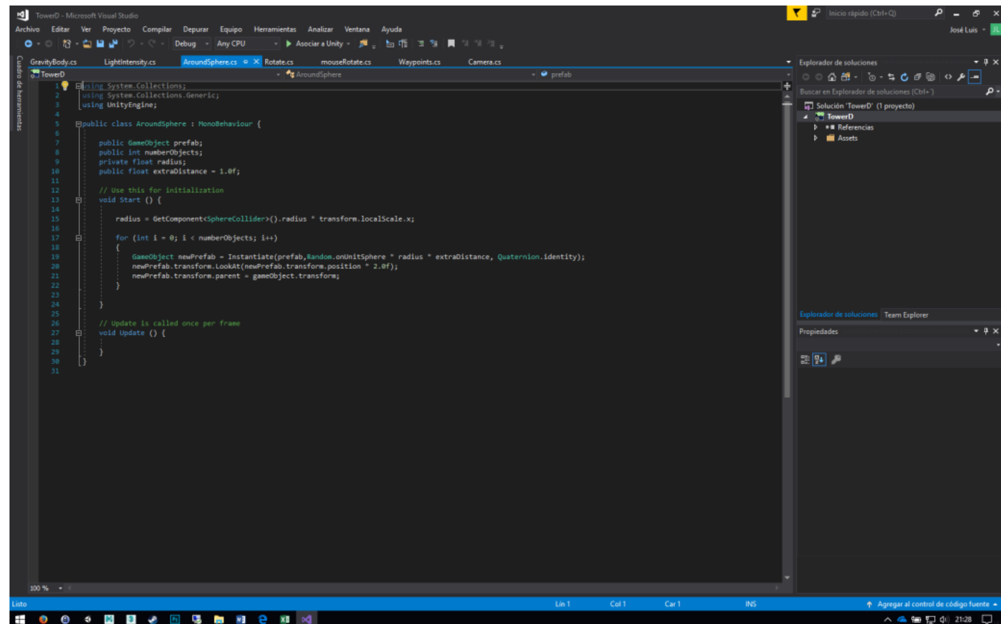

Figura 8 - Entorno de trabajo de Visual Studio Community 2017

Soporta multitud de lenguajes de programación como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP, ASP.NET, … Aunque en este caso se le dará uso exclusivo al lenguaje de programación C#.

Unity 3D da soporte a trabajar con Visual Studio Community o MonoDevelop. A efectos prácticos podremos conseguir los mismos resultados, si he escogido Visual Studio Community es porque me gusta más la interfaz gráfica y fluidez a la hora de trabajar. Así, que es básicamente a elección gustos personales.

Tanto Visual Studio Community como MonoDevelop son totalmente gratuitos.

3. Planificación del proyecto

El proyecto está pensado para abarcar todas las características que posee un videojuego y siendo consciente de que el proyecto lo presentaré en septiembre, me propongo tener como orden de prioridad un nivel jugable, menú principal y mapa de elección de nivel. Aunque me gustaría continuar el proyecto y poder desarrollar un videojuego completo, teniendo en cuenta las dificultades que esto conlleva siendo una única persona para noviembre o diciembre.

Actividades	Inicio	Fin	Días
Idea	15/04/2017	22/04/2017	7
Información, documentación Unity, videos	23/04/2017	13/05/2017	20
Información posible software modelado y audio	23/04/2017	30/04/2017	7
Primeros bocetos de niveles y personajes	30/04/2017	01/05/2017	1
Pruebas técnicas con objetos primarios y programación	01/05/2017	06/05/2017	5
Sistema de gravedad y waypoints	06/05/2017	20/05/2017	14
Modelado, diseño y desarrollo del primer nivel (Planeta)	16/05/2017	11/06/2017	26
Música de fondo del primer nivel	11/06/2017	13/06/2017	2
Modelado, diseño e implementación de un enemigo	13/06/2017	24/06/2017	11
Animación del enemigo	24/06/2017	01/07/2017	7
Modelado, diseño e implementación de una torreta	01/07/2017	08/07/2017	7
Diseño menú principal	08/07/2017	15/07/2017	7
Implementación del mapa de niveles	15/07/2017	20/07/2017	5
Añadir niveles de dificultad	20/07/2017	23/07/2017	3
Modelado, diseño e implementación de más enemigos y torretas	23/07/2017	23/10/2017	92
Modelado, diseño y desarrollo de más niveles	23/09/2017	23/11/2017	61
Publicación en Steam Early Access	23/11/2017	26/11/2017	3
		TOTAL:	**278**

Tabla 3 - Planificación del proyecto por fecha de inicio, fin y días.

En cuanto a añadir más enemigos no es para todos los niveles por igual, sino, que habrá enemigos comunes para todos los mapas y otros exclusivos de cierto/s nivel/es.

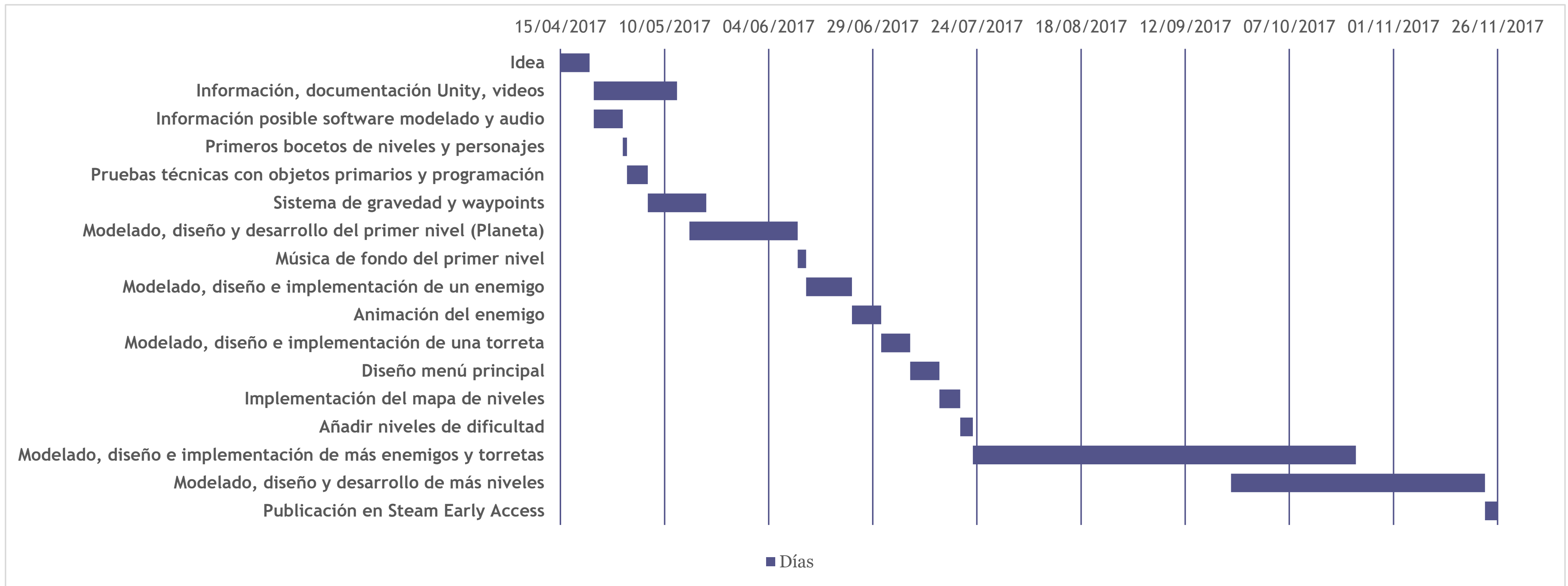

Gráfico 2 - Diagrama de Gantt del proyecto.

4. Desarrollo del videojuego

4.1. Problema principal

El primer problema con el que nos encontramos al trabajar con Unity 3D (y todos los motores de videojuegos que conozca), es que las físicas están inicializadas de forma que exista una fuerza de gravedad en la coordenada Y, ya que está pensado para trabajar sobre un plano.

Como en este caso vamos a tener que desarrollar un cuerpo (planetoide) que atraiga a los demás objetos hacia él, lo más conveniente es deshacernos de esta preconfiguración inicial.

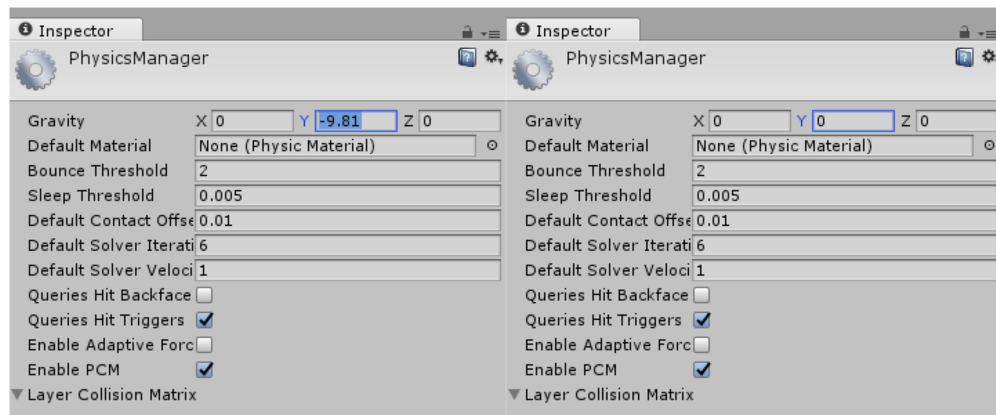

Figura 9 – La tierra no es plana. Nuestro planetoide tampoco.

De esta forma a partir de ahora, todos los objetos físicos que tengamos dentro de la vista del nivel permanecerán intactos a no ser que le apliquemos fuerza.

4.2. Crear un planetoide

Crear un planetoide a efectos prácticos si lo hacemos lo más básico posible no es muy complicado. Simplemente es crear una esfera en el escenario, pero el problema real es que este atraiga a los demás objetos y que además estos se mantengan "de pie". Es decir, una persona que se encontrara en el polo norte estaría totalmente invertida a otra que estuviera en el otro punto opuesto en el polo sur.

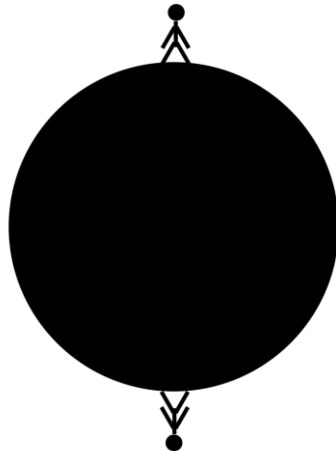

Figura 10 - Las personas no deben caminar con su cabeza.

Para ello, se les asigna a los objetos físicos el cuerpo celeste por el que deben ser atraídos y se les programa una fuerza hacia la posición central de este. Y a su vez la rotación de estos objetos en base a su eje Y local (conocido como Vector3.up en Unity) alineándolos sobre la posición central del cuerpo celeste.

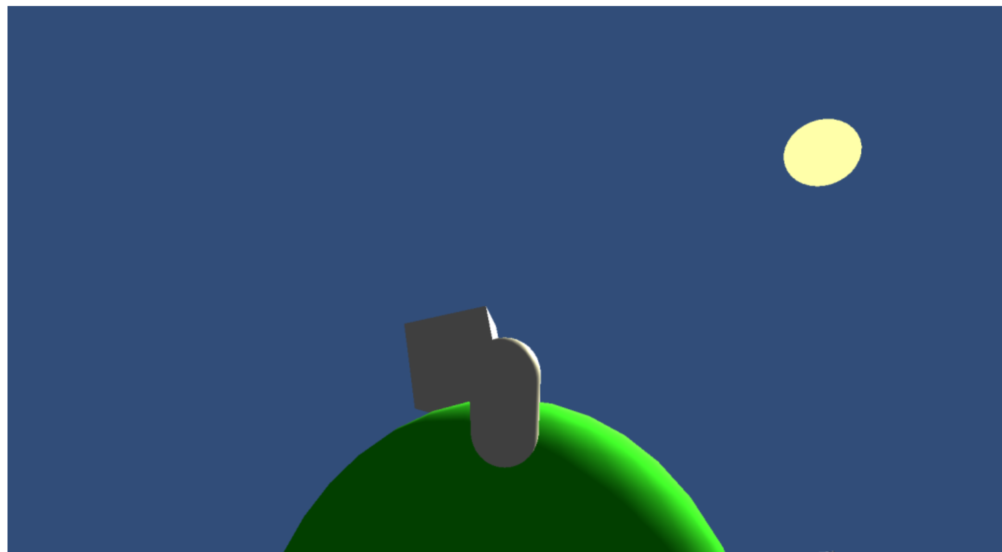

Figura 11 - Prueba técnica sobre la atracción de un planetoide.

4.3. Sistema de Waypoints

Los enemigos tienen que caminar siguiendo una ruta desde un punto inicial A hasta otro punto B, pasando por puntos intermedios (ya que el camino no será regular, tendrá curvas, …). Para ello, tendremos que crear objetos no visibles en el resultado final, para que nuestros enemigos se dirijan consecutivamente uno detrás de otro hacia los puntos donde se encuentran.

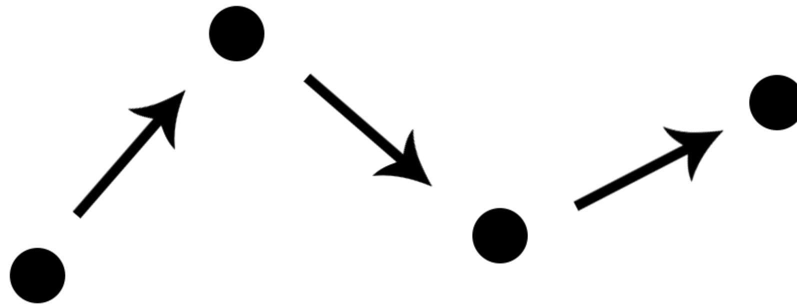

Figura 12 - Sistema de Waypoints

La dificultad de este proceso es que la 'cara' del enemigo debe mirar hacia el siguiente punto y esto provoca que hay que hacer una rotación del objeto cuando ya contábamos con rotaciones de este sobre el planetoide, por lo que hay que normalizar la rotación de los dos procesos, y programar y hacer varias pruebas hasta que el resultado sea el adecuado.

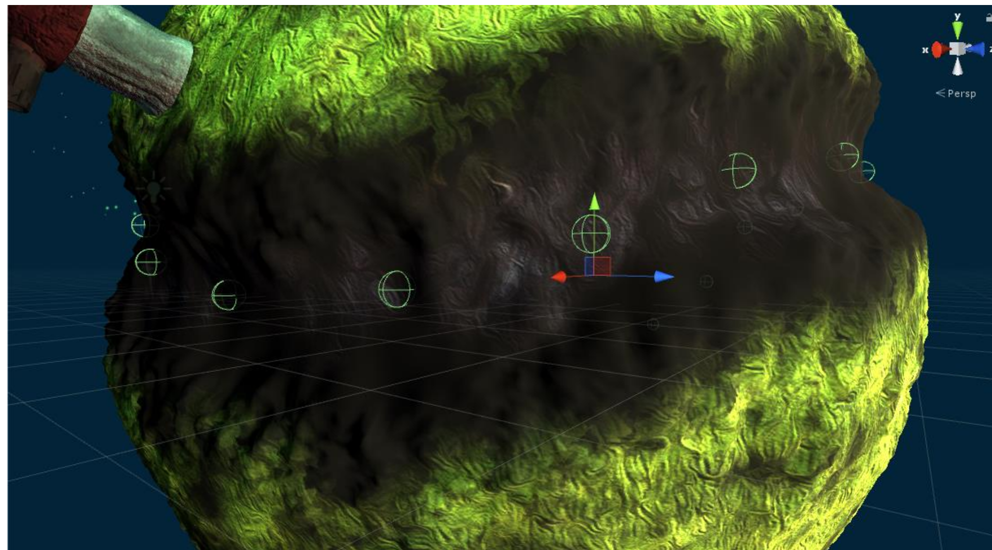

Figura 13 - Objetos haciendo de Waypoints sobre el camino.

4.4. Modelado de un enemigo

El proceso de modelado de un personaje puede ser más o menos sencillo dependiendo del nivel de detalle que queramos conseguir. En mi caso voy a desarrollar un modelo con un nivel de detalle elevado y una vez conseguido reducirle este nivel para que pueda ser cargado óptimamente en un videojuego, pero utilizando técnicas que permitan que el modelo parezca que sigue teniendo un nivel de detalle alto.

4.4.1. Base del modelo

En primera estancia lo que hago es crear una base poligonal sin ningún detalle, en el que se identifiquen las partes del cuerpo (Torso, cabeza, brazos, piernas, …), utilizando para ello 3ds Max.

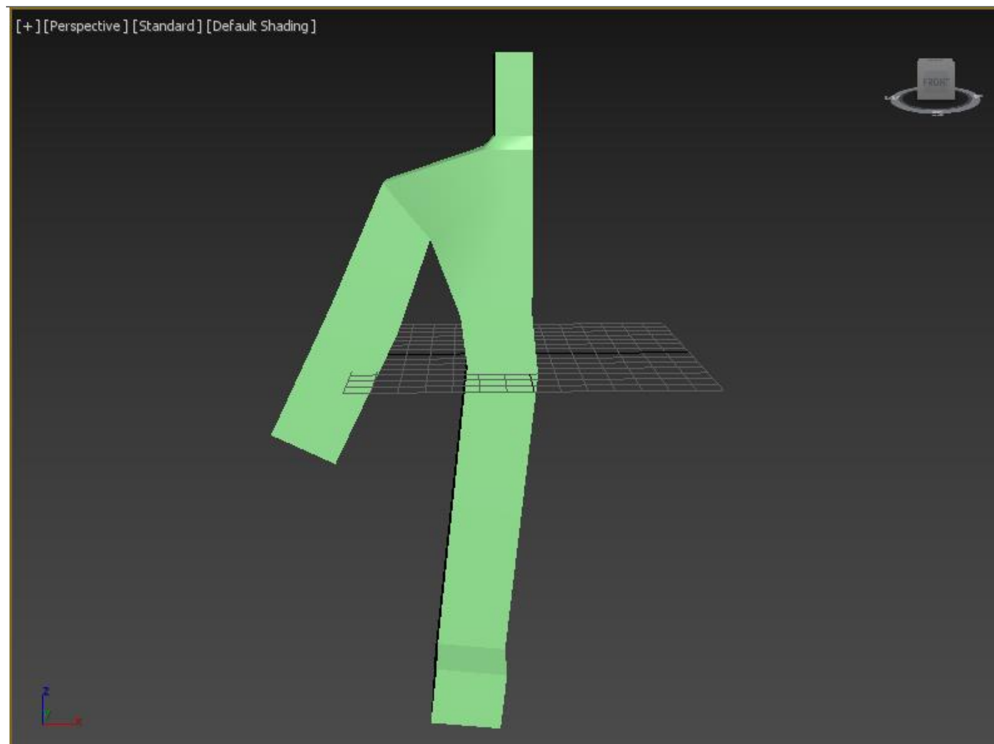

Figura 14 - Base poligonal de un enemigo con la que trabajar.

Para facilitar el trabajo, solo desarrollaremos una parte del cuerpo, ya que la otra la podremos duplicar en modo espejo. El modelo tiene que ser lo más sencillo posible ya que de eso me encargo después.

4.4.2. Modelado a alto nivel

En el siguiente paso importo la base en Mudbox. Aquí esculpo y voy añadiendo niveles de detalle al modelo hasta ir consiguiendo los detalles. En este proceso se pinta el modelo y se le dan detalles de piel, rasguños, arrugas, … que se crean convenientes.

Figura 15 - Modelado en alto detalle

En este estado el modelo cuenta con 132.352 polígonos. Esto quiere decir que, si contáramos con los polígonos del planetoide, sol, decoración, cientos de enemigos, … sobrecargaría el videojuego de forma innecesaria.

4.4.3. Generar mapa de normales

Normalmap o mapa de normales es una textura que generamos para dibujar los detalles que actualmente tiene el modelo en alto nivel de detalles, para que los tenga en forma de textura el mismo modelo reduciendo su número de polígonos. Esto no es una textura que se multiplica en escala de grises sobre la textura principal, sino que indica también profundidad, por lo que cuando el modelo reciba luz está reflejara como si realmente tuviera profundidad.

Figura 16 - Mapa de normales de un enemigo.

De momento no hago uso de este mapa de normales, sino que lo guardo y haré uso de él en Unity 3D.

4.4.4. Reducción de polígonos

A continuación, reduzco el número de polígonos del modelo al mínimo quedándose en tan solo 2.068 polígonos (64 modelos en este estado equivaldrían a 1 de alto nivel).

Figura 17 - Modelo a bajo nivel.

Ahora exporto este modelo en una extensión que entienda Unity 3D y paso a importarlo en el mismo.

4.4.5. Modelo a bajo nivel con mapa de normales

El siguiente paso es importar el modelo que cuenta con un nivel de polígonos y detalles bajo.

Figura 18 - Modelo a bajo nivel en Unity 3D.

Los detalles de este modelo son bajos. Lo que hago es añadirle el mapa de normales y cambia su aspecto totalmente, dejando un modelo con un aspecto parecido a 64 veces más de su nivel de polígonos consumiendo muchísimos menos recursos.

Figura 19 - Modelo a bajo nivel en Unity 3D con mapa de normales aplicado.

En este caso, el videojuego no es del estilo ‘shooter’ y no vamos a tener el modelo cerca de nosotros por lo que el resultado va a ser mucho mejor, puesto que la distancia de nuestra visión respecto del enemigo es mayor.

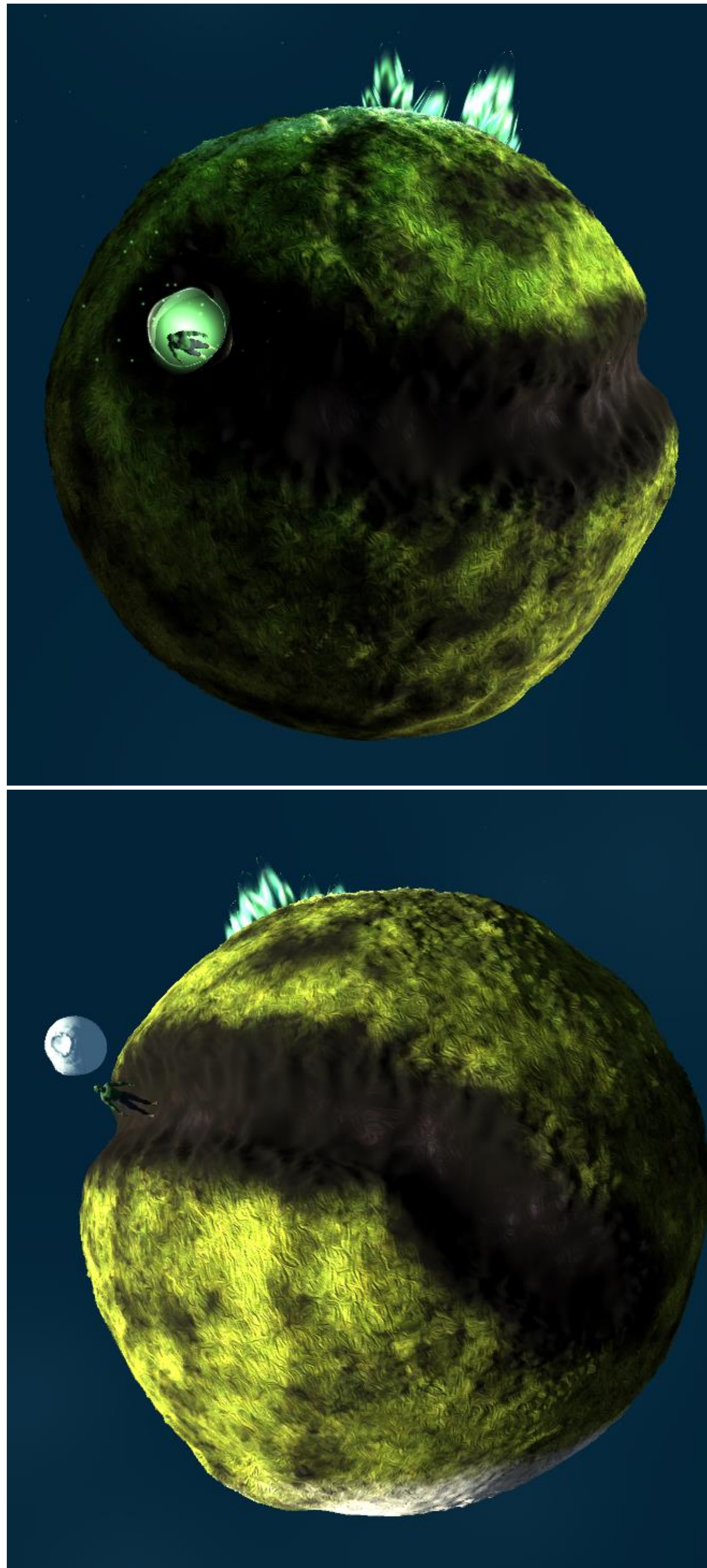

Figura 20 - Previsualización del enemigo en Unity 3D. Planetoide modelado con Mudbox.

4.4.6. Animación del enemigo

Para ello, exporto el modelo en bajo nivel de Mudbox a 3ds Max. Y le aplico un esqueleto de tipo bípedo al modelo. En este proceso se tiene que ajustar el esqueleto a la maya del modelo, y una vez ajustado lo que se hace es aplicar el filtro Skin (Piel), ajustando el esqueleto al modelo.

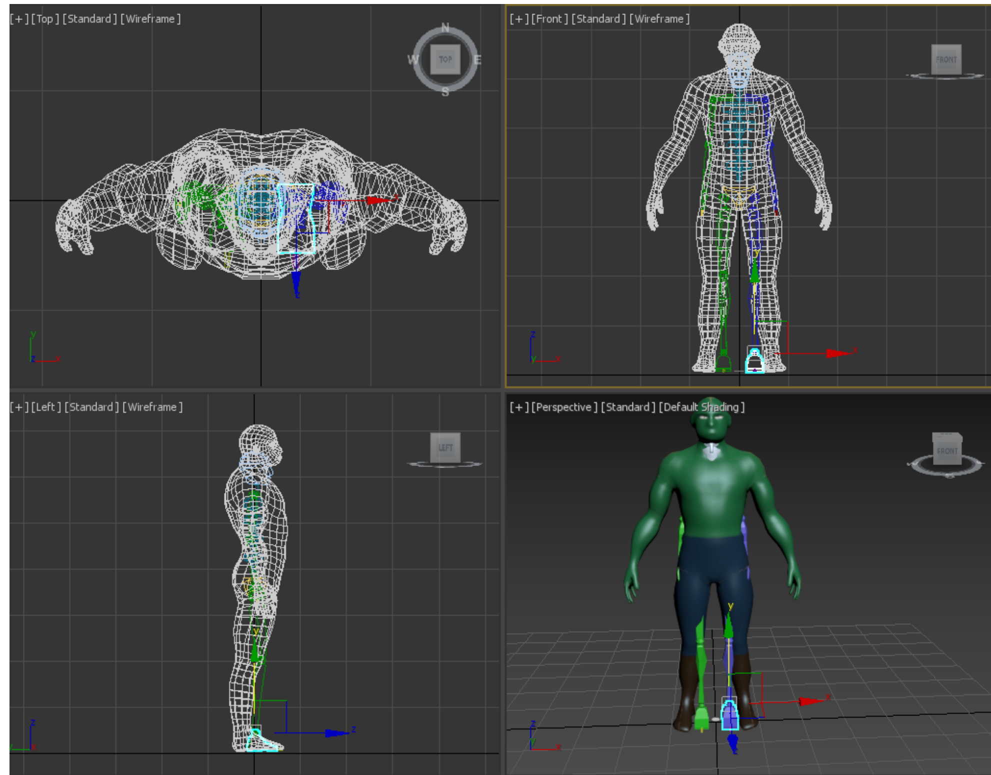

Figura 21 - Aplicando un esqueleto bípedo al modelo.

A partir de aquí, se pueden hacer varias cosas, o crear las animaciones en 3ds Max, usar un software externo para aplicar animaciones predefinidas, o desde el mismo Unity 3D. Yo me he decantado por la última opción.

5. Texturizado

5.1. Texturizando una esfera

Cuando se trabaja con esferas, en este caso un planetoide, nos encontramos con el problema de que al texturizarla (mapearla), se produce un punto de unión de la textura creando un efecto de absorción.

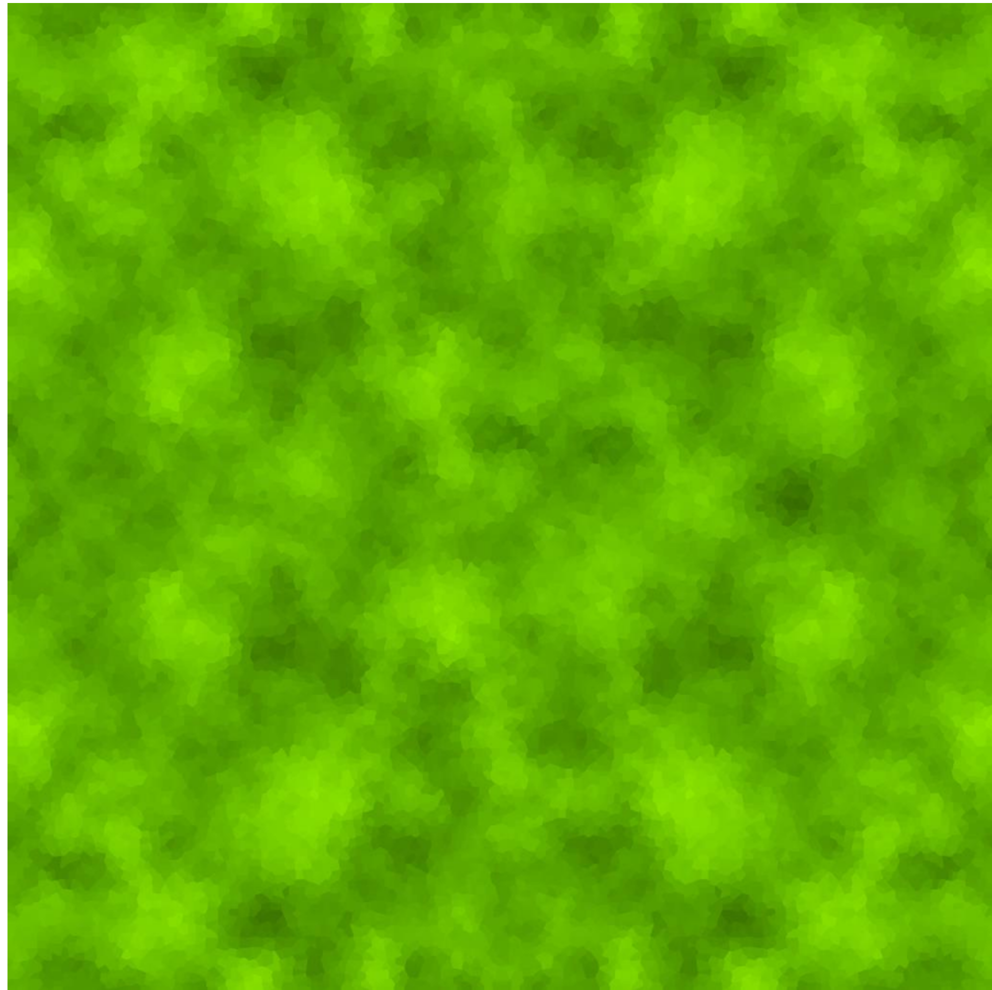

Figura 22 - Textura realizada con photoshop.

Si realizamos y aplicamos una textura en la que coincidan sus extremos con el lado opuesto, servirá para planos y cubos, pero si lo hacemos sobre la esfera no será suficiente.

Figura 23 - Textura aplicada sobre una esfera.

Para conseguir que se vea correctamente lo que hago es utilizar el filtro de coordenadas polares en Photoshop. Con esto solo conseguimos uno de los polos, para que el polo opuesto también se vea correctamente en hacer la textura el doble de alta y duplicar la parte superior a la parte nueva inferior, y voltearla verticalmente.

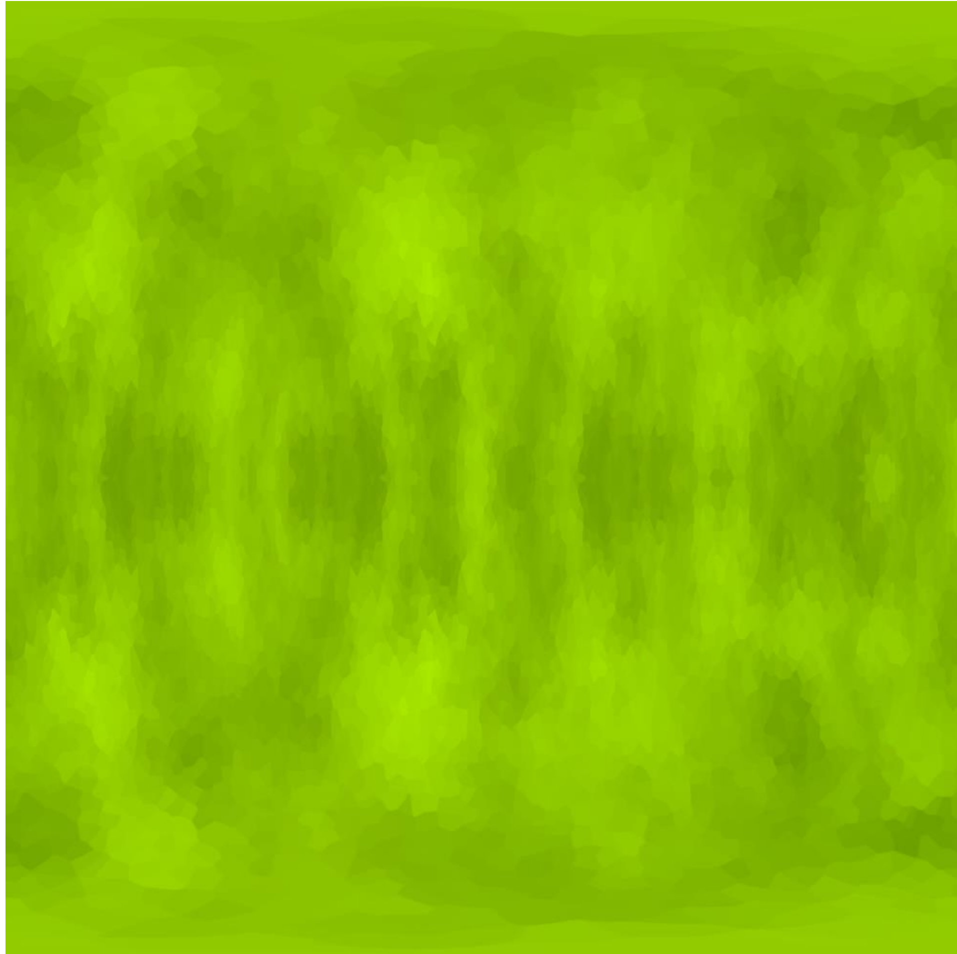

Figura 24 - Textura con el filtro de coordenadas polares aplicado.

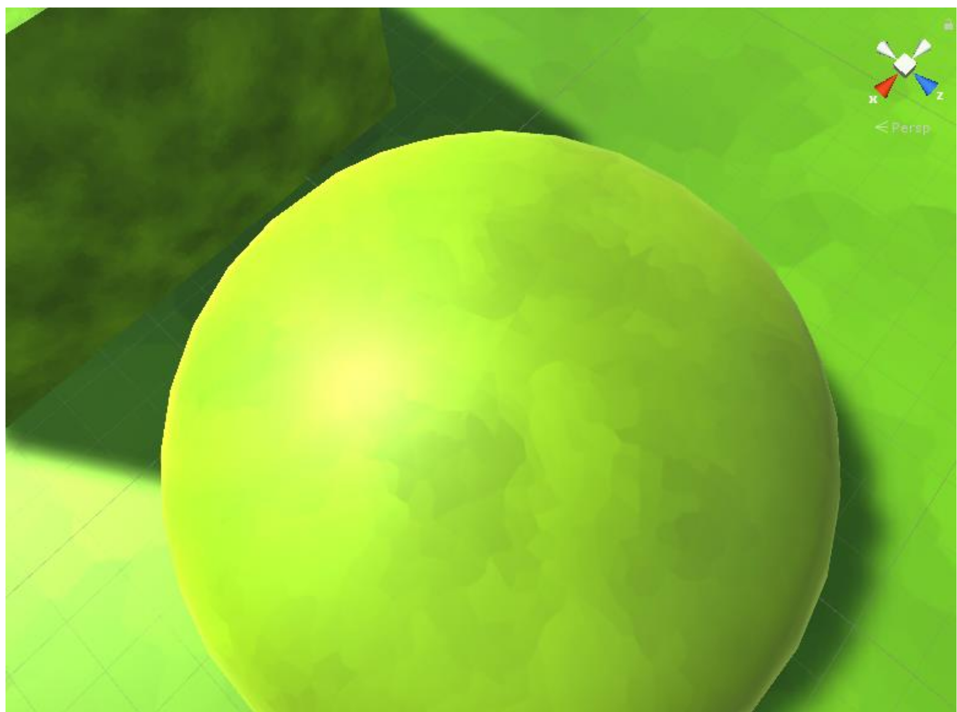

Figura 25 - Textura con el filtro de coordenadas polares aplicado a una esfera.

La lógica empleada en este caso ha sido el mapamundi de la tierra, donde sabemos que está 'estirado' el mapa por los polos.

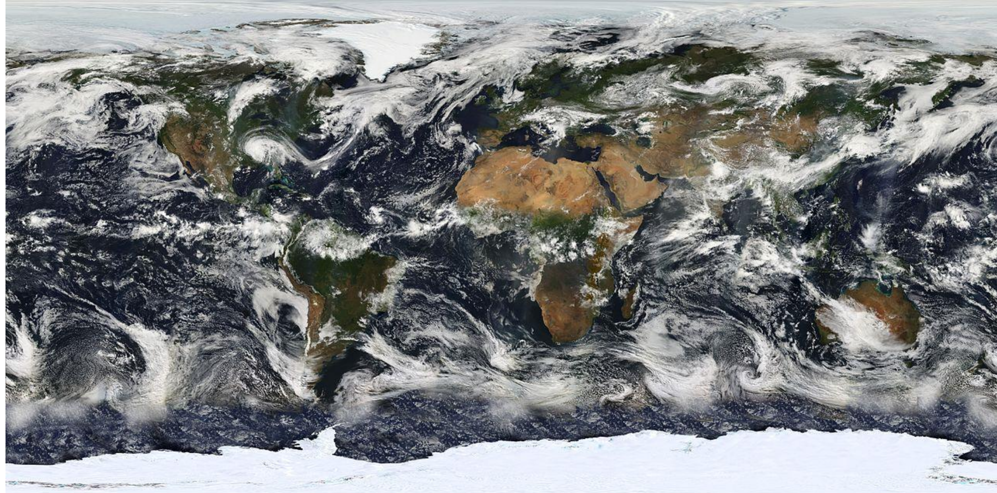

Figura 26 - Mapamundi de la Tierra. Fuente: es.wikipedia.org

Esta técnica servirá para el desarrollo de los planetoides, y estrellas cercanas decorativas del escenario y evidentemente cualquier otro objeto esférico que se pueda presentar.

6. Sistema de párticulas

6.1. Definición

A efectos prácticos es la representación de efectos tales como gases, humo, llamas, … que están presentes en la naturaleza. Se suelen presentar generalmente en pantalla mediante la acumulación de varios sprites, que son texturas en blanco y negro, donde la cantidad de negro representa el nivel de transparencia o mediante texturas con transparencia directamente. También se pueden utilizar objetos, aunque no es lo habitual.

6.2. Nubes

Para conseguir este efecto, primero me propongo realizar una atmosfera, ya que voy a usar está atmosfera para alojar en su forma esférica las nubes, y que estas vayan rotando a la par de la atmosfera. Esto también me va a ayudar a crear el cielo, aunque el jugador no será consciente de que existe realmente puesto que no va a poder salir del planetoide, pero en caso de poder hacerlo se comprobaría que el espacio es oscuro.

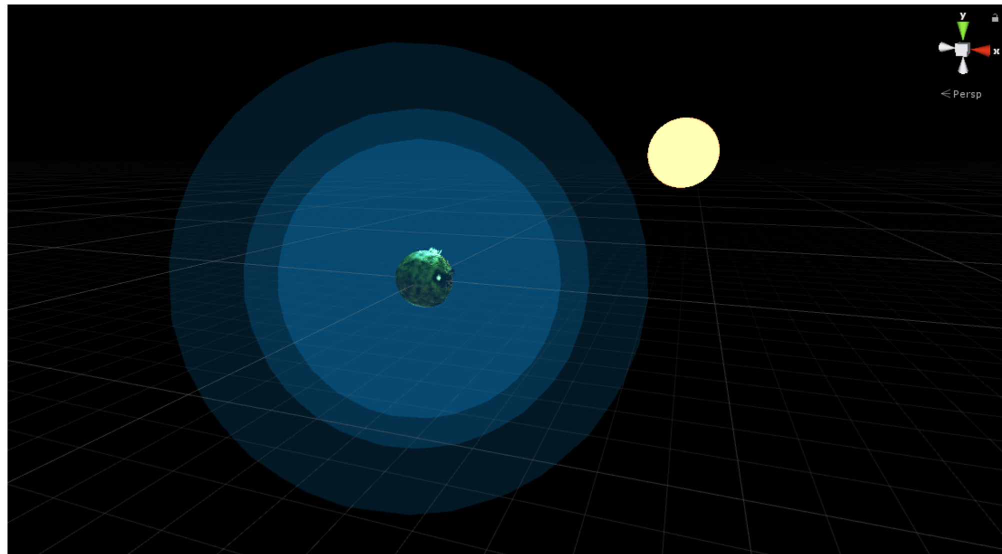

Figura 27 - Atmosfera del planetoide.

He eliminado la luz direccional que inserta Unity 3D por defecto y el Skybox (El paisaje de fondo, cielo, …), ya que por defecto Unity 3D te prepara la escena situándote ya dentro de un entorno. En cuanto a la iluminación es nuestro propia estrella la que emite la luz en tiempo real, mientras rotamos alrededor de él.

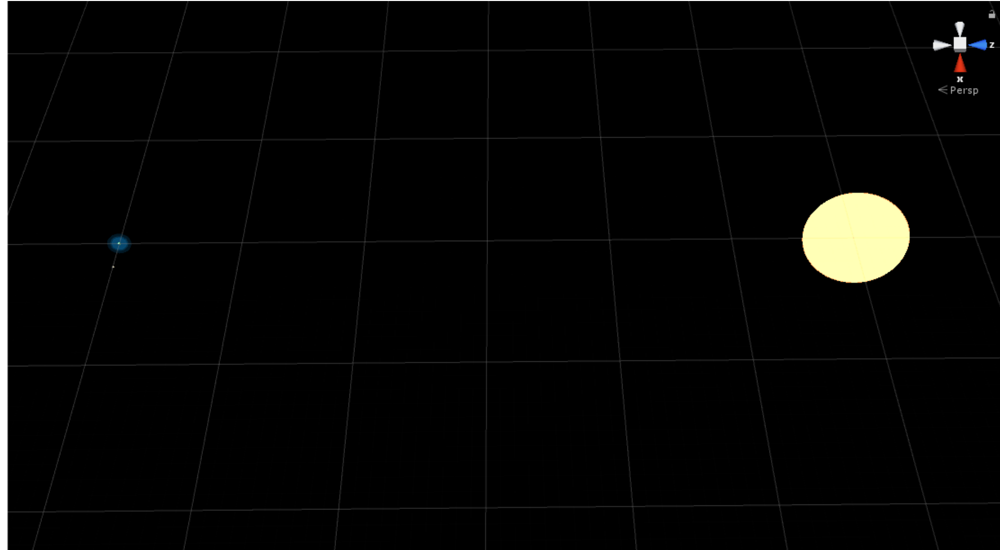

Figura 28 - Sistema planetario, compuesto por el planetoide, la estrella y un satélite.

He realizado un Script que me permite generar aleatoriamente objetos sobre la superficie de una esfera. Teniendo esto en cuenta entendemos que nuestra esfera es la atmosfera y nuestros objetos son las nubes.

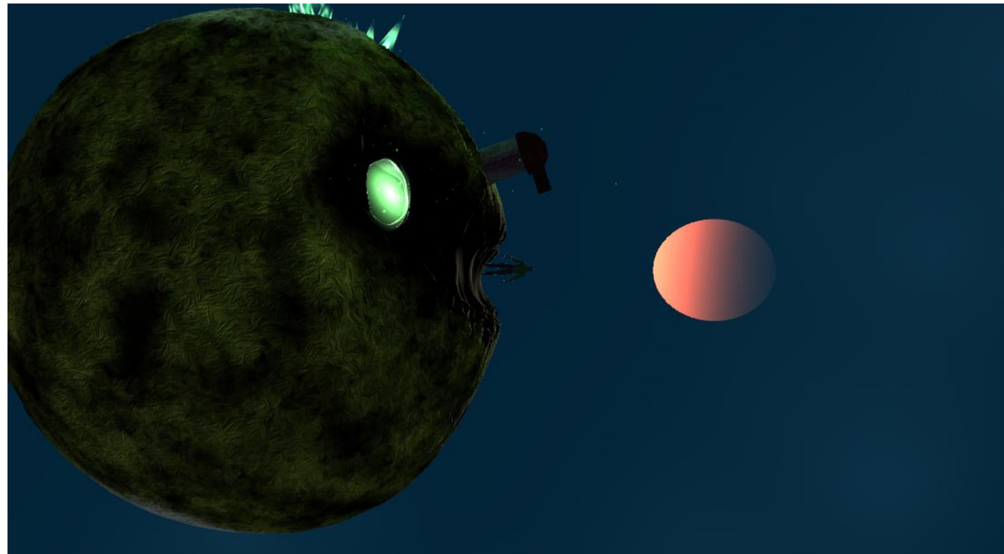

Figura 29 - Nuestra estrella ocultándose detrás de las nubes. Parte oscura del planetoide.

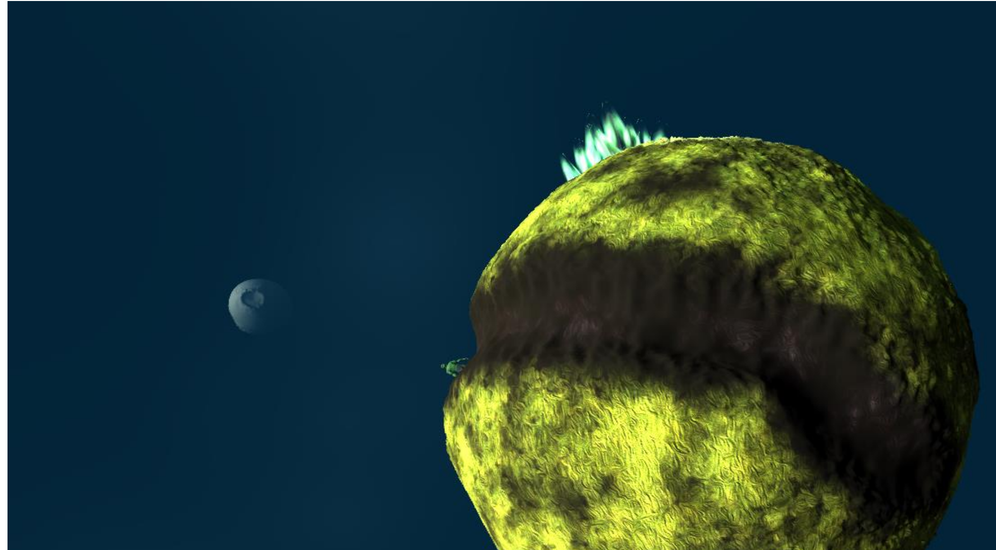

Figura 30 - Satélite ocultándose entre las nubes. Parte iluminada del planetoide.

6.3. Partículas emanando de la luz

Esto resulta más sencillo puesto que he superpuesto varios sistemas de partículas parecidos, naciendo desde una forma de cono. Les he dado diferente tamaño, color, distancia y velocidad y aplicado un Script de rotación sobre si mismas respecto su eje Y local para darles efecto de remolino.

Figura 31 - Partículas emanando de la luz.

7. Programación

7.1. Rotación del planetoide

Refiriéndome a la que realiza el jugador respecto la cámara y no la rotación natural del planeta respecto la estrella. Para ello, creo una esfera sin maya más grande que el planetoide para que no interfieran los demás objetos.

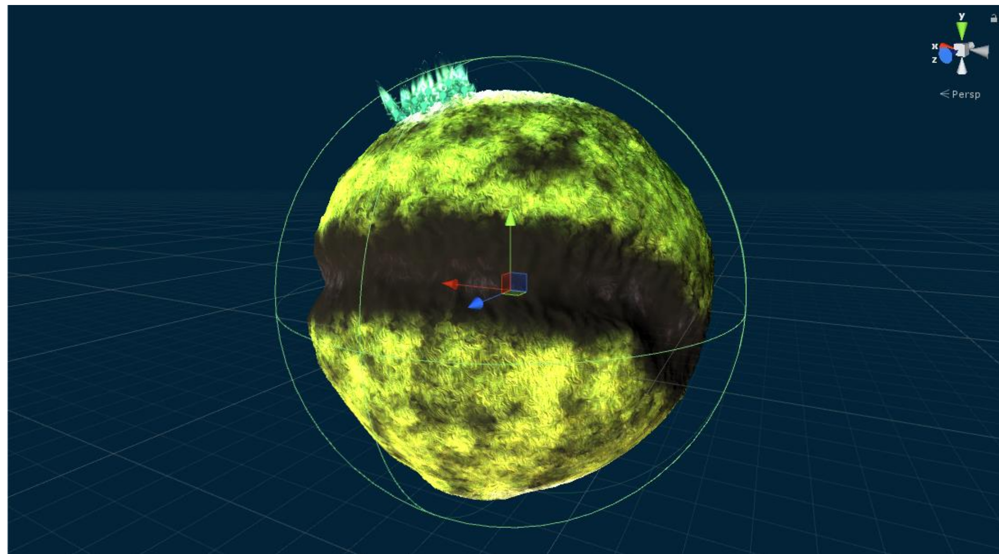

Figura 32 - Esfera al rededor del planetoide.

Esta esfera será padre de la cámara. Así, cuando rotemos la esfera la cámara seguirá la misma rotación. Para que la esfera rote creo un script que, al situarse por encima de la esfera, al mantener clicado el ratón sobre ella y arrastrar detecte las posiciones de desplazamiento del ratón y gire la esfera junto la cámara a la misma velocidad solamente de forma lateral.

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

public class mouseRotate : MonoBehaviour {

    private float sensitivity;
    private Vector3 mouseReference;
    private Vector3 mouseOffset;
    private Vector3 rotation;
    private bool isRotating;

    // Use this for initialization
    void Start () {
        sensitivity = 0.4f;
        rotation = Vector3.zero;
    }

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

        if (isRotating)
        {
            mouseOffset = (Input.mousePosition - mouseReference);
            rotation.y = -(mouseOffset.x) * sensitivity;
            //rotation.x = -(mouseOffset.y) * sensitivity;
            transform.Rotate(rotation);
            mouseReference = Input.mousePosition;
        }
    }

    void OnMouseDown()
    {
        isRotating = true;
        mouseReference = Input.mousePosition;
    }

    void OnMouseUp()
    {
        isRotating = false;
    }


}
```

Figura 33 - Código que nos permite rotar la esfera que controla la cámara.

7.2. Rellenar una esfera de objetos

Este script lo desarrollé en principio para llenar el planetoide de hierba, flores, … cuando el planetoide carecía de detalle y no quería que quedara pobre, pero al desarrollar mi aprendizaje sobre Mudbox me permitió realizar un planeta mucho más detallado y este script lo rechacé.
Sin embargo, lo pude volver a utilizar al crear el sistema de partículas de las nubes, ya que no dejan de ser objetos generados aleatoriamente sobre una esfera (atmósfera).

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class AroundSphere : MonoBehaviour {
6
7      public GameObject prefab;
8      public int numberObjects;
9      private float radius;
10     public float extraDistance = 1.0f;
11
12     // Use this for initialization
13     void Start () {
14
15         radius = GetComponent<SphereCollider>().radius * transform.localScale.x;
16
17         for (int i = 0; i < numberObjects; i++)
18         {
19             GameObject newPrefab = Instantiate(prefab,Random.onUnitSphere * radius * extraDistance, Quaternion.identity);
20             newPrefab.transform.LookAt(newPrefab.transform.position * 2.0f);
21             newPrefab.transform.parent = gameObject.transform;
22         }
23
24     }
25
26     // Update is called once per frame
27     void Update () {
28
29     }
30 }
```

Figura 34 – Código que nos permite añadir objetos aleatoriamente sobre una esfera.

7.3. Intensidad de luz variable

He creado sobre el planetoide objetos que emiten luz propia, como es la balsa de la vida, donde nacen los enemigos, o conjunto de piedras transparentes en los polos. Para conseguir un efecto más dinámico en el que las luces se van atenuando y volviendo a emitir más luz he desarrollado un script que divide por cuatro el valor de la intensidad de luz del objeto como valor mínimo y el valor inicial como máximo. De forma que gradualmente vaya reduciéndose la emisión hasta llegar al valor mínimo y una vez alcanzado volviendo al valor máximo.

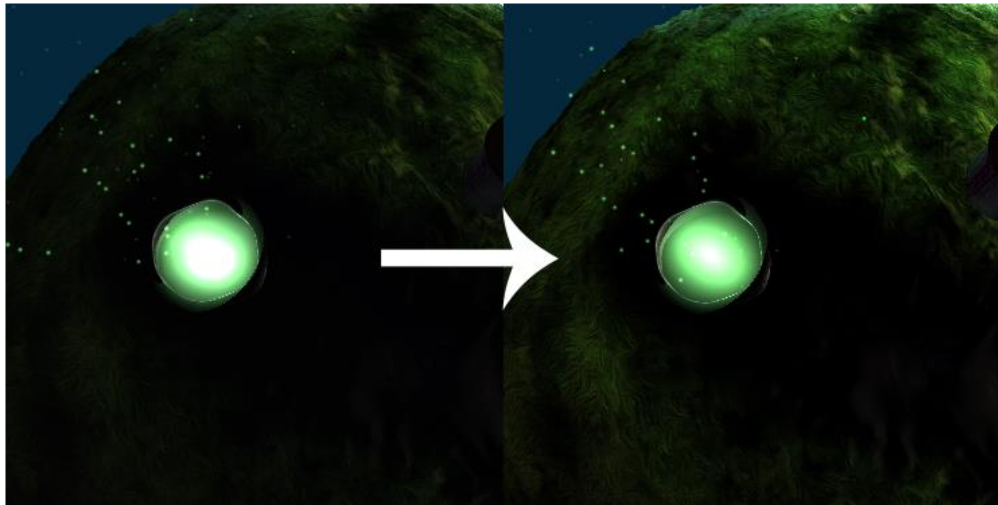

Figura 35 - Cambio de intensidad de la iluminación.

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

public class LightIntensity : MonoBehaviour {

    private float startIntensity;
    private bool min;
    public float level;

    // Use this for initialization
    void Start () {
        min = false;
        startIntensity = GetComponent<Light>().intensity;
    }

    // Update is called once per frame
    void Update () {
        if (!min)
        {
            GetComponent<Light>().intensity -= level;
            if (GetComponent<Light>().intensity <= startIntensity/4)
                min = true;
        }
        else
        {
            GetComponent<Light>().intensity += level;
            if (GetComponent<Light>().intensity >= startIntensity)
                min = false;
        }

    }
}
```

Figura 36 - Código que nos permite cambiar la intensidad de la luz.

7.4. Sistema de waypoints

Para la implementación de este script lo que hago es generar un array de objetos (waypoints) y se lo asigno al enemigo. De esta manera lo que hago es que se desplace hacia el primer objeto del array y en el momento que colisiona con él, le digo que se desplace hacia el siguiente y así sucesivamente hasta llegar el último que es la meta y destruyo el enemigo. Este script deberá reducir nuestra puntuación aún no implementado cuando el enemigo llegue a la meta.

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

public class Waypoints : MonoBehaviour {

    public float speed = 5.0f;
    public Transform[] targets;
    Transform target;
    int i;


    // Use this for initialization
    void Start () {

        i = 0;
        target = targets[0];

    }

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

        if (i < targets.Length)
        {
            Vector3 pos = ProjectPointOnPlane(transform.up, transform.position, target.position);
            transform.LookAt(pos, transform.up);

            //transform.LookAt(target);
            transform.position += transform.forward * speed * Time.deltaTime;
        }

    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.GetComponent<Transform>() == target)
        {
            if (i < targets.Length - 1)
            {
                i++;
                target = targets[i];
            }
            else
            {
                Destroy(gameObject);
            }

        }
    }

    Vector3 ProjectPointOnPlane(Vector3 planeNormal  , Vector3 planePoint , Vector3 point ) {
        planeNormal.Normalize();
        float distance = -Vector3.Dot(planeNormal.normalized, (point - planePoint));
        return point + planeNormal* distance;
    }

}
```

Figura 37 - Código que nos permite seguir un recorrido de waypoints.

7.5. Cámara centrada al planetoide

Este es el script más sencillo de todos. Solo consta de una línea, y es para centrar la cámara al centro del planetoide.
También se podría utilizar para encarar un objeto hacia otro.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Camera : MonoBehaviour {
6
7       public Transform target;
8
9       // Use this for initialization
10      void Start () {
11          transform.LookAt(target);
12      }
13
14      // Update is called once per frame
15      void Update () {
16
17      }
18  }
```

Figura 38 - Código que nos permite centrar la cámara en el planetoide.

7.6. Planetoide y objetos atraídos

Estos dos scripts son la piedra angular donde gira el sistema de gravedad del videojuego.
En primer lugar, tenemos el script asociado al planetoide, que se encarga de que todos los objetos que asignemos que sean atraídos por el planetoide, mantengan su vector Y local alineado al centro del planetoide y se le aplique una fuerza en este sentido, atrayéndolos al centro del planeta.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class GravityAttractor : MonoBehaviour
5  {
6
7      public float gravity = -9.8f;
8
9
10     public void Attract(Rigidbody body)
11     {
12         Vector3 gravityUp = (body.position - transform.position).normalized;
13         Vector3 localUp = body.transform.up;
14
15         body.AddForce(gravityUp * gravity);
16
17         body.rotation = Quaternion.FromToRotation(localUp, gravityUp) * body.rotation;
18     }
19 }
```

Figura 39 - Código que nos permite atraer a otros objetos a nuestra posición.

Por otro lado, tenemos el script asociado a los objetos atraídos, en los que simplemente enviamos nuestro objeto a realizar las funciones del primer script.

```
1  using UnityEngine;
2  using System.Collections;
3
4  [RequireComponent(typeof(Rigidbody))]
5  public class GravityBody : MonoBehaviour
6  {
7
8      GravityAttractor planet;
9      Rigidbody rigidbody;
10
11     void Awake()
12     {
13         planet = GameObject.FindGameObjectWithTag("Planet").GetComponent<GravityAttractor>();
14         rigidbody = GetComponent<Rigidbody>();
15
16         rigidbody.useGravity = false;
17         rigidbody.constraints = RigidbodyConstraints.FreezeRotation;
18     }
19
20     void FixedUpdate()
21     {
22         planet.Attract(rigidbody);
23     }
24 }
25
```

Figura 40 - Código que nos permite asociar el objeto para ser atraído al planetoide.

8. Música

Esta es la parte más difícil de redactar, puesto que no puedo representar la música (de forma acústica) como tal en la documentación. Por lo que voy a explicar la lógica que he seguido para crearla.
El ejemplo en concreto será la música de fondo del primer nivel. En el que inicializo con percusión suave (Bongos) y completados dos minutos inicializo la percusión que estará presente en toda la pista hasta el final.
En el minuto 5 inicializo dos efectos de percusión de platillo dando pie al bajo (que consta de dos samples). Tanto los efectos de percusión que da comienzo en el minuto 5 como los dos samples del bajo se irán repitiendo hasta el final.
La batería se inicializará dos minutos y medio después de la percusión presente hasta el final acompañándola también hasta el final y siendo esta la que acabará la música de fondo.
Esto obviamente se comprobará mejor en la defensa.

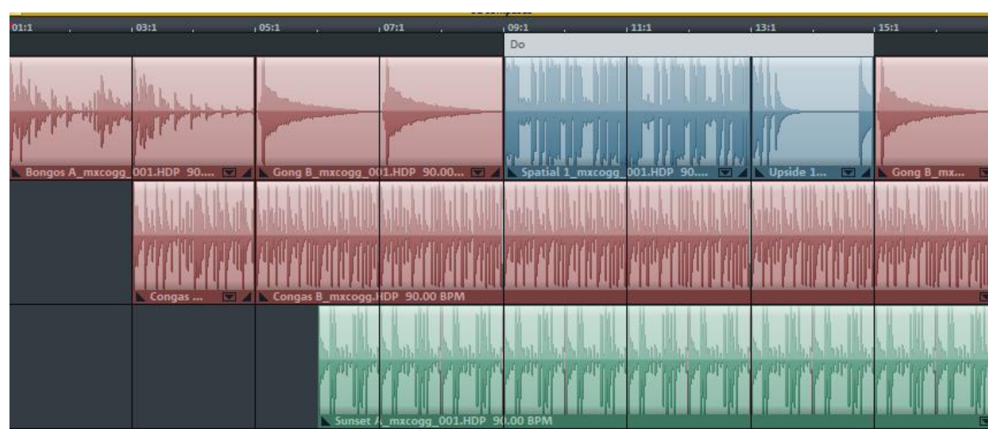

Figura 41 - Representación gráfica de las pistas al inicio de la música del primer nivel.

9. Trabajo futuro

Me gustaría cumplir con lo propuesto en el diagrama de Gantt aunque me lleve más tiempo del previsto, pero antes de hacer más niveles, me gustaría tener uno con todas las opciones bien pulidas para que no se me repliquen los errores en otros niveles que pudiera realizar.
Los puntos a tener en cuenta teniendo en cuenta esto serían:

- Diseño menú principal.
- Implementación del mapa de niveles (Tengo pensado usar el mismo sistema planetario creado. Es decir, hacer un juego de cámaras en el que te alejas y acercas a los planetoides para jugar y no una escena aparte).
- Añadir niveles de dificultad.
- Modelado, diseño e implementación de más enemigos y torretas.
- Modelado, diseño y desarrollo de más niveles (Sería interesante añadir otros sistemas planetarios, pero claro es algo que tendré que valorar por el tiempo que puede conllevar).
- Publicación en Steam Early Access.

10. Conclusiones

El tener que desarrollar varios puntos de un videojuego me ha hecho darme cuenta lo esencial que es formar un equipo de personas que se especialicen en puntos concretos del videojuego, más hoy en día en el que el nivel de detalle y resoluciones son más altas. Incluso hay estudios que cuentan con personas que solo se especializan en el desarrollo de partículas.

Creo que es importante conocer todas las partes que se compone un videojuego, así siempre puedes saber de las posibilidades de desarrollo y tiempo que requiere cada trabajo, y como no, poder conseguir trabajar en cualquiera de ellos o poder elegir en un momento determinado que es lo que más te gusta.

El principal problema que me encontré aparte del de abarcar varios artes (ya me lo imaginaba que me lo encontraría),es la idea del videojuego. Puesto que puedes empezar a desarrollar y darte cuenta que la idea no era la correcta. Esto me ha pasado, y a mitad proyecto he tenido que cambiar el tipo de videojuego que estaba desarrollando de tipo plataforma (Cuando tenía el personaje principal, historia, y scripts,…) en el que el personaje saltaba pasando entre planetoides a un estilo de Tower Defense, manteniendo el sistema de planetoides y gravedad. También ha tenido su beneficio, puesto que en cierta manera me ha permitido innovar dentro de este tipo de juegos.

Agradezco los conocimientos previos durante la carrera que me han permitido desarrollar parte del videojuego, como puede ser la programación de scripts y por supuesto la asignatura de introducción a la programación de videojuegos. Esto me ha facilitado ampliar conocimientos y poder utilizarlos en un mundo que me gusta. El de los videojuegos.

11. Bibliografía

[1] Unity tutorials.
URL: https://unity3d.com/es/learn/tutorials

[2] How to Walk Around a Sphere (Walk on Planet) in Unity 3D.
URL: https://www.youtube.com/watch?v=cH4oBoXkE00

[3] How to Instantiate Objects in a Circle.
URL: https://forum.unity3d.com/threads/how-to-instantiate-objects-in-a-circle.10693/

[4] Mario galaxy in unity3d platformer engine mario galaxy style.
URL: https://www.youtube.com/watch?v=soeKTEksWF4

[5] Procedural quadtree planet in Unity.
URL: https://www.youtube.com/watch?v=aV6e5eHTfUI

[6] Creating a Dynamic Sky in Unity.
URL: http://www.eastshade.com/creating-a-dynamic-sky-in-unity/

[7] Mudbox for Beginners.
URL: https://www.youtube.com/watch?v=H3qZwt5l3xE

[8] 3ds Max 2018 - Tutorial for Beginners.
URL: https://www.youtube.com/watch?v=cb3_X9grdmc

[9] Página web de Unity 3D.
URL: https://unity3d.com/es

[10] Página web de 3ds Max.
URL: https://www.autodesk.es/products/3ds-max/overview

[11] Página web de Mudbox.
URL: https://www.autodesk.com/products/mudbox/overview

[12] Página web de MAGIX Music Maker.
URL: http://www.magix.com/es/music-maker/

[13] Página web de Adobe Creative Cloud.
URL: http://www.adobe.com/es/creativecloud.html

[14] Página web de Steam:
URL: http://store.steampowered.com/