View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Diposit Digital de Documents de la UAB



# AKONE- Aliens Know One.

# Diseño e implementación de un videojuego tipo shooter en Unity

Levani Makharashvili Tvauri

**Resumen** — En el mundo de los videojuegos cabe destacar la unión de tres factores muy importantes; el diseño del juego, el arte y la programación. En mi TFG (Trabajo Final de Grado) he optado por desarrollar un prototipo del juego de género shooter en la cual participen estos tres factores. La evolución del proyecto ha sido notable en cuanto a la fase de aprendizaje, investigación y conexión entre distintas herramientas que han facilitado a unificar todo el proceso. El trabajo de un ingeniero informático frente a este tipo de proyecto es sin duda la adaptación al medio y la capacidad de resolución de problemas demostrando el uso de su ingenio. AKONE ha sido el resultado.

**Palabras claves** — Shooter, AKONE, Unity, 3DsMax, Photoshop, game, game design, game concept, script, Power Up, rigging

**Abstract** — In the world of video games it should be noted that the union of three factors: game design, art, and programming; is very important. As a final thesis, it has been decided to develop a playable demo in which those three factors are very present. The evolution of the project has been notable in terms of these phases: learning, investigation and connection between various tools that made all the composition much easier. The work of and engineer among these type of projects is without a doubt the adaptation to the environment and the capacity to solve problems showing the use of its inventiveness. AKONE is the final result.

**Index Terms** — Shooter, AKONE, Unity, 3DsMax, Photoshop, game, game design, game concept, script, Power Up, rigging

## 1 INTRODUCCIÓN

La creación de un videojuego comercial es una tarea compleja y actualmente requiere meses de trabajo en equipos de decenas de personas, con mucha experiencia y con un gran presupuesto. Hasta el juego comercial más sencillo hoy en día exige un buen conocimiento en diferentes disciplinas, si se pretende llevar a cabo, además de un gran esfuerzo y voluntad de aprender constante.

La motivación principal a la hora de realizar este proyecto ha sido mi amor incondicional por los videojuegos y el propósito de crearlos. Decidí cursar la carrera de ingeniería informática para poder llevar a cabo este propósito y después de mucho sacrificio y esfuerzo han convergido todas las condiciones necesarias para conseguirlo.

AKONE es el fruto del esfuerzo de diseño, arte y programación de un juego realizado durante un periodo de aprendizaje limitado (Trabajo Final de Grado) en el cual se ha creado la história de AKONE, un planeta dominado por una raza de alienígenas que gobierna gracias a su poder increíblemente destructivo.

Por todo ello, el objetivo principal de este proyecto es la creación y desarrollo de un videojuego de modalidad *shooter* en tercera persona mediante el motor gráfico de Unity agregando un toque musical y colorido. En este artículo se explica con detalle la evolución del mismo, desde el diseño del juego, el diseño de personajes y modelado en 3D, su texturizado, la realización de animaciones y la implementación del *game play*.

Se ha estructurado el proyecto en estas tareas:

- Diseño del *game concept* (história, personajes, entorno gráfico, ambientación).
- Diseño de los personajes, niveles del juego (diseño vectorial en 3 dimensiones con 3DsMax 2019 y creación de texturas mediante Photoshop CC).
- Animaciones del personaje (utilizando 3DsMax 2019 y la herramienta Mixamo de Adobe).
- Implementación del *game play* mediante Unity (importación de los modelos previamente diseñados y texturizados), que incluye:
  - Creación de un árbol de animaciones para el personaje principal y sincronización del movimiento.
  - Creación de *scripts* para el movimiento el personaje principal, sitema de disparos y lógica del juego.
  - Creación e implementación de la IA básica para los enemigos del juego.
  - Diseño de una interficie de usuario inmersiva.
- Pruebas de test y corrección de errores.

En la sección 2 explicaremos el diseño del juego, en la sección 3 como se ha realizado el arte 3D del juego, en la sección 4 se

————————————————

- *E-mail de contacto: Levani.makharashvili@gmail.com*
- *Mención realizada: Intenieria del Software.*
- *Trabajo tutorizado por: Enric Martí Gódia (Departamento de Ciencias de la Computación.*

explica todo lo referente a *Game Play* en Unity, en la sección 5 comentamo los resultados obtenidos, en la sección 6 se menciona propuestas de mejora y por ultimo los agradeciminetos. En los anexos se muestra un boceto de AKONE, su diseño, imágenes del juego finalizado y su diagrama de clases.

## 2. Diseño del juego

AKONE es un juego de modalidad tipo *shooter* ambientado en un planeta imaginario en el cual reina la *Madre AKONE*, un ser malvado y destructivo que utiliza seres vivos de otros planetas obligándoles a combatir en una batalla contra la vida y la muerte con simple afán de divertirse, y para ello, los teletransporta a su planeta y los fusiona con sus AKONEs (prototipos de alienígena) para potenciarlos y enfrentarlos en sus escenarios recreados telepáticamente.

Como experiencia de juego, se pretende que el jugador, entre en un estado de euforia por sus colores, aceleración por los sonidos, alta concentración por los disparos y movimiento de los enemigos ya que aboslutamente todo estará en simbiosis con la música.

El tipo de musica que predomina en AKONE es *dubstep*, tanto la música de ambiente, como los disparos de cada uno de los personajes y la interacción con los Power Ups transmitiendo una inmersión total dentro del juego.

Se estima una duración de 180 segundos de partida facilitando al jugador un temporizador decrementativo mostrado en la interfaz de usuario.

Cabe destacar que el juego va dirigido a un público de 12 a 30 años con la intención de generar un juego totalmente competitivo en modalidad *shooter*.

Como personaje principal de este videojuego tenemos a AKONE, un ser dual compuesto por una parte humanoide y un AKONE (alienígena); ha de transmitir astucia, inteligencia, velocidad, agresividad y picardía; cabe destacar que el personaje tendrá un aspecto aerodinámico, con armas por brazo con una gama de colores verdes y azulados con pequeños detalles en rojo. (figura 1)

*Figura 1: AKONE. Logo + personaje principal.*

AKONE debe superar una fase de pruebas impuesta por su majestad la *madre* AKONE y para ello, debe sobrevivir en un periodo finito de tiempo.

En el campo de pruebas, AKONE se va a encontrar con una serie de enemigos llamados MegaAKONEs, seres duales compuestos por un elemento y un AKONE (figura 2), que tendrán como objetivo matar a cualquier AKONE; además de enemigos, en la fase de pruebas se hallarán flotando cubos de energía (*Power Ups* en adelante) que tendrán la capacidad de potenciar al personaje haciendo que active sus poderes ocultos gracias a la música, y así poder superar la fase de pruebas sin ningún problema.

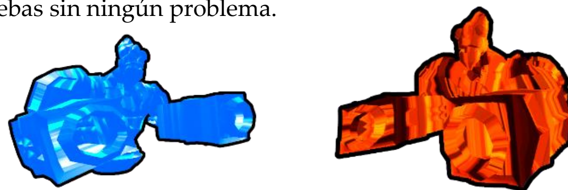


*Figura 2: MegaAKONE_Aqua y MegaAKONE_Gamma.*

### 2.1 Plataformas de Desarrollo

Como principal plataforma de desarrollo se ha utilizado Unity, motor gráfico desarrollado por Unity Technologies el cual nos facilitará toda la implementación de nuestro videojuego ofreciéndonos herramientas como visor de animaciones, entorno de desarrollo de programación en C# incorporando Visual Studio de Microsoft, utilizando OpenGL (en Windows, Mac OS y Linux), Direct3D (sólo en windows), y soportando también el mapeado de relieve, de reflejos y cartográfico entre otras cosas.

Como plataformas para el desarrollo del arte hemos utilizado 3DsMax2019 de Autodesck para todo el proceso de diseño vectorial en 3 dimensiones y *Rigging* de los personajes y objetos. Se ha optado por 3DsMax2019 ya que ofrece una licencia de prueba gratuita para estudiantes de 3 años de duración y es muy utilizado en la indústria del videojuego. Para el texturizado se ha utilizado Adobe Photoshop CC. Y para las animaciones de los personajes se ha utilizado Mixamo de Adobe.

Una vez explicado el concepto de videojuego que queremos desarrollar, procedemos a explicar el trabajo realizado para la creación de AKONE tanto en arte como en *game play*.

## 3. Arte 3D

Los objetos y escenarios que se han modelado para AKONE son los siguientes:

- AKONE como personaje principal.
- MegaAKONE_Aqua como enemigo nº1.
- MegaAKONE_Gamma como enemigo nº2.
- Power Ups en forma de cubo.
- Teletransportadores en forma de cuadrado.
- Escenario dividido en 3 secciones; un escenario de inicio, un escenario neutral y el escenario del campo de pruebas.
- Iluminación.
- Interfaz de usuario.

A modo de ejemplo, explicaremos cómo hemos modelado el personaje principal.

### 3.1 Modelado del personaje principal

El personaje principal se ha diseñado siguiendo unos dibujos (adjuntos en el anexo del documento) realizados en el documento de diseño. El procedimiento ha sido el siguiente:

1 - Se ha establecido un plano de referencia con el dibujo de AKONE con el objetivo de poder dar volumen a la imagen en 2D del personaje.

2 - Modelado del personaje principal.

3 - El siguiente paso ha sido texturizar; darle color a nuestro modelo y para ello se ha tenido que desenglosar el personaje en una malla plana (2 dimensiones) gracias a la herramienta UVLayaout que ha facilitado el proceso. Esta malla plana se ha exportado a Photoshop CC y se ha procedido a pintar.

4 - Se ha aplicado esta malla coloreada a nuestro modelo en 3d y para ello dentro del entorno de 3DsMax se ha creado un material (*type checker*) asegurándonos de que lo que hemos coloreado con Photoshop se ha adherido correctamente a nuestro personaje (figura 3).

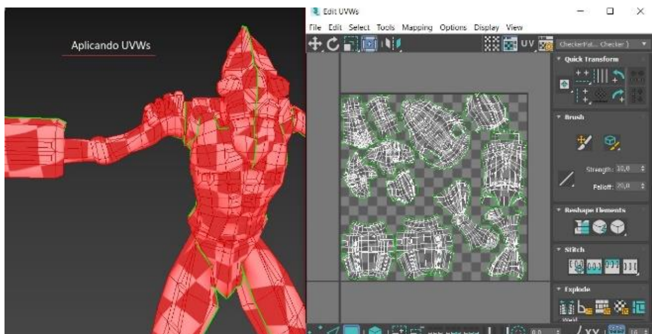



*Figura 3: Modelo preparado para su utilización en Unity.*

Tanto los enemigos como los Power Ups han seguido el mismo procedimiento a la hora de ser diseñados.

## 4. Game Play

Nuestro objeto principal es AKONE y éste ha de poder moverse por el escenario, interactuar con Power Ups y tener una repercusion a la hora de enfrentarse a sus enemigos (disparar y recibir disparos).

AKONE, se representa como un objeto de Unity (*Game Object),* una figura en 3D básica (cubo, esfera, cápsula, etc.) la cual se podrá editar y así poder tratar las físicas y el movimiento de lo que más adelante será AKONE.

### 4.1 Game Objects

El Game Object es el tipo de objeto más importante en Unity. Cada objeto que se vaya a crear en nuestro juego va a ser un Game Object. No obstante, los Game Objects no hacen nada por sí mismos, necesitan propiedades especiales antes de que puedan convertirse en un personaje, un ambiente, o un efecto especial. Pero cada uno de estos objetos puede hacer diferentes cosas. Si cada objeto es un Game Object, ¿cómo podríamos diferenciar un Power Up interactivo de un cuarto estático? ¿Qué hace que estos Game Objects sean diferentes del uno al otro?

Los Game Objects son contenedores, pueden guardar las diferentes piezas que componen un personaje, una luz, un árbol, un sonido, o lo que uno quiera construir; cada una de estas piezas se llama *Component,* y para acceder a ellos, usaremos el *inspector,* propiedad del editor de Unity que nos permite la asignación de multiples piezas (o compontentes) a los Game Objects de Unity, pudiendo ser tanto scripts como elementos de audio u otras piezas.

Dependiendo del objeto a crear, agregaremos una serie de components diferentes para cada Game Object. Unity dispone de varios componentes integrados y con ayuda de los scripts que vayamos generando, van a ir cogiendo forma los Game Objects, así le daremos forma a AKONE y a todo lo que aparecerá en el scenario.

A continuación se procede a explicar cada objeto (*game object*) del juego, sus atributos y características así como su implementación.

AKONE antes de ser implementado, se tratará como un Game Object *Cápsula* de Unity. Este objeto tendrá varios componentes, entre los cuales destacaremos el *RigidBody*[1], *Capsule Collider*[2] y *Animator.*

Paralelamente trabajaremos con el Game Object de tipo *Camera,* con el que representaremos una vista de AKONE desde su espalda, dando una perspectiva *ThirdPersonCamera,* que nos definirá la cámara del juego.

Implementaremos el controlador de la cámara que nos ofrece Unity (*CameraController*) para que siga a nuestro personaje y para ello generaremos un script encargado de enfocar a nuestra cápsula; todas las variables harán referencia a vectores de coordenadas X e Y y se considerará como *target* a la cápsula que más adelante será reemplazada por el modelo en 3D de AKONE.

### 4.2 Game Objects - Movimiento

Para generar el movimiento de nuestro Game Object cápsula, crearemos un script de movimiento (*CharacterController*).

En el script se declaran y usan variables de movimiento (aceleración, salto, ejes de coordenadas vertical y horizontal) para más adelante tratar con ellas y generar funciones que nos permitirán desplazarnos tanto en el eje vertical (forward, back, left, right) como horizontal (jump).

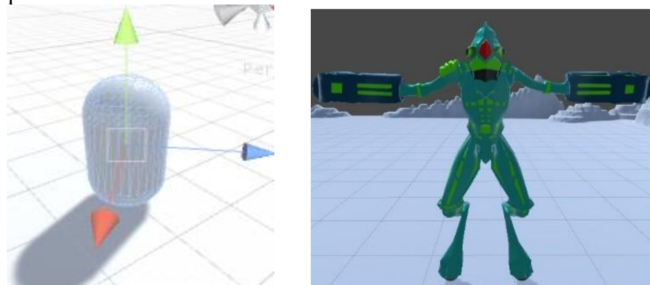
Una vez conseguido el movimiento de la cápsula, se procede a implementar la funcionalidad del salto, la cual actua sobre la aceleración y la gravedad seguido de los ejes de coordenadas vertical y horizontal permitiéndonos saber si el personaje está colisionando con el suelo o no.

Hasta ahora, tenemos un Game Object representado por una cápsula con dos tipos de controladores, el del movimiento y el

[1]RigidBody: Componente de Unity que permite la interacción y uso de físicas en un objeto, pudiendo así darle una cierto realismo a nuestros objetos pudiendo modificar su masa como su gravedad e incluso la detección de colisiones.

[2]Collider: Componente de Unity que define en un objeto la propiedad de colisionar; un collider es invisible y recubre el objeto, por lo tanto si dos objetos tienen este componente (personaje y suelo) el personaje no atravesará el suelo.

de la cámara; ha llegado el momento de sustituir la cápsula por AKONE (nuestro objeto modelado) tal y como se aprecia en las figuras 4(a) y 4(b) para poder tener un personaje y proceder a animarlo más adelante.



*Figra 4(a): Cápsula.* *Figura 4(b): AKONE.*

### 4.3 Game Objects - Salto

A diferencia del movimiento del personaje principal AKONE, cuando hablamos del salto hemos de interpretar qué es lo que sucede cuando nuestro persnaje salta; ¿se debe de aplicar una gravedad? ¿la posición del personaje varía?

Para contestar a la primera pregunta cabe decir que Unity interpreta el salto de la siguiente manera: Nuestro objeto player necesita una nueva variable de tipo booleana para describir si el personaje se encuentra en el suelo, es decir, encima de algún objeto como un plano. Si la respuesta es afirmativa, entonces no estamos saltando. En cambio si no está encima de un plano o un objeto, interpretaremos que está saltando.

Si el usuario quiere que su personaje salte, aprieta la barra espaciadora. Esto quiere decir que vamos a recibir un *input* por teclado y hacer que pase algo. En este caso, si detectamos que se ha pulsado la barra espaciadora, aplicamos físicas a nuestro personaje, un desplazamiento del eje Y de coordenadas multiplicado por la gravedad, con valor de 7 en lugar de 9,8m/s para simular que no estamos en la Tierra si no en el planeta AKONE. Todo esto se puede realizar gracias a las físicas que Unity nos ofrece; librería denominada *UnityEngyne* implementada desde *UnityEngyne.PhysicsModule.*

En respuesta a la segunda pregunta, se contempla que el personaje esté desplazándose y quiera realizar un salto en movimiento; la funcionalidad del salto en este caso será similar. Haremos además de un desplazamiento en el eje Y, un desplazamiento en el eje X y en el eje Z multiplicados por la gravedad.

Hasta el momento hemos generado un personaje en 3D que se mueve y salta, pero no es lo único que queremos. Nuestro objetivo es transmitir una serie de movimientos a nuestro personaje y para ello, hay que realizar animaciones.

### 4.4 Game Objects - Animaciones

Unity cuenta con un gestor de animaciones integrado que se basa en una linea temporal de *frames* y nos permite tratar con objetos previamente diseñados con una base de *Rigging* (un esqueleto generado dentro del objeto que nos permite articularlo sin deformarlo) y generar animaciones fotograma por fotograma.

Unity, distingue entre *Transiciones* y *Blend Trees* (árboles de mezcla). A pesar de que ambos se utilizan para animar cualquier tipo de objeto, se suelen utilizar para diferentes tipos de situación.

Las *Transiciones* sirven para hacer una transición suave de un estado de animación (*Animator State*) a otro durante un periodo de tiempo determinado. Las transiciones son específicas como parte de una máquina de estados de animación (*Animator State Machine,* figura 6), que vienen siendo el cambio entre una animación y otra, por ejemplo que un personaje pueda respirar o balancearse ligeramente cuando esté en reposo, caminar cuando se le indique, saltar, etc.

Los *Blend Trees* se utilizan para mezclar animaciones de forma suave al incorporar una parte de todas las animaciones en diferentes grados. En otras palabras, que tenga sentido una secuencia de movimiento. Por ejemplo, si nuestro personaje está corriendo y el jugador para, el efecto de la animación pasa de correr a parado (*run to idle*).

Se muestra en las figuras 5a y 5b la visualización de la animación de correr.

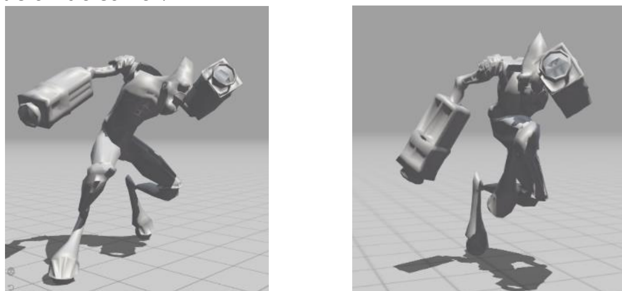


*Figura 5a: Animación de correr* *Figura 5b: Animación de correr.*

Una vez generadas las animaciones, se procederá a explicar el proceso de interconexión desde su *Animator State Machine.* En el diagrama de la (figura 6) se muestra qué es lo que pasa cuando entramos en un bucle de animación (*Entry*, caja verde del diagrama):

· Como primer bloque o estado, nos encontramos con *Movement*, la representación del movimiento del personaje compuesta por 5 animaciones: *Idle, Left State, Right State, Back* y *Walking*. Se ha representado en color naranja y es lo que previamente se ha explicado, un *Blend Tree* (figura 6) seguida de la figura 7 representando lo que *Movement* contiene.

· A su vez, se muestran unas flechas blancas que tienen dos direcciones, una para entrar en el siguiente bloque de animaciones y la otra para volver. Estas flechas son *Transiciones*.

· El cambio de estados puede variar en función del evento por teclado que utilice el jugador. Si está quieto o se mueve, permanecerá en el estado de *Movement*, si aprieta la barra espaciadora para saltar, cambiará de la animación de

*Movement* a *Jump_Static* (siendo ésta la animación de salto básica del personaje), si el usuario aprieta la tecla *shift*, se cambiará de estado de *Movement* a *Running* y si está en *Running* y quiere saltar, pasará al estado de *Running_Jump*.

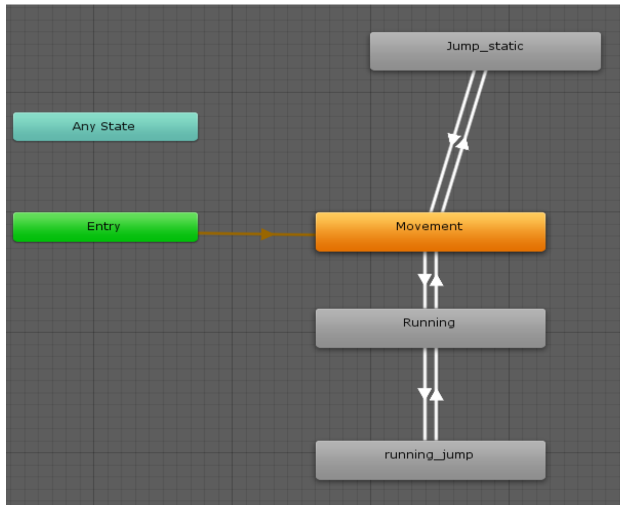



*Figura 6: Animator State Machine.*

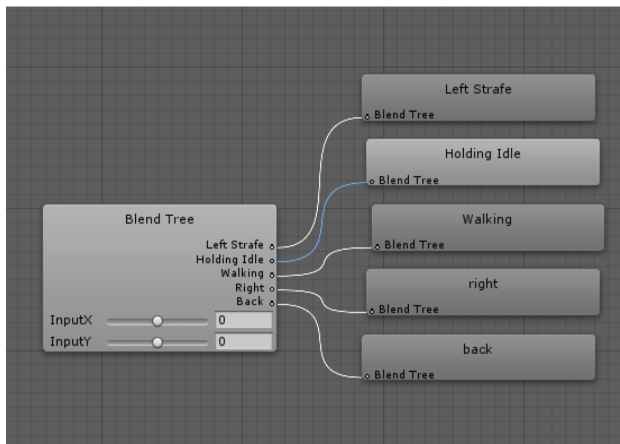



*Figura 7: Blend Tree de Movement.*

Analicemos con más detalle lo que está sucediendo en nuestro *Blend Tree* de *Movement (figura 8)*: Como se muestra en la (parte superior azul (1)) se aprecian los parámetros de entrada Input X e Input Y. El movimiento asignado (2) hace referencia a cada una de las animaciones y finalmente la visualización de la animación (3). En el que se observa un eje de coordenadas X y un eje de coordenadas Y, debido a que Unity interpreta cada animación como estado y lo controla mediante una asignación numérica la cual le ayuda a saber en qué animación se encuentra respecto al evento de entrada por teclado del usuario; en este caso, hemos asignado para el estado de *idle* la posición (X = 0, Y = 0), el (estado neutral de nuestro objeto AKONE). Si el usuario selecciona la tecla (*W*), se produce una transición de estado de *idle* a *Walking* debido a que *walking* tiene la posición (X = 0, Y = 1) permitiendo a Unity identificar que se ha producido un desplazamineto y en consecuencia, cambie de animación. Lo mismo sucece con *Left State* (X = -1, Y = 0), *Right* (X = 1, Y = 0) y por ultimo *Back* (X = 0, Y = -1).

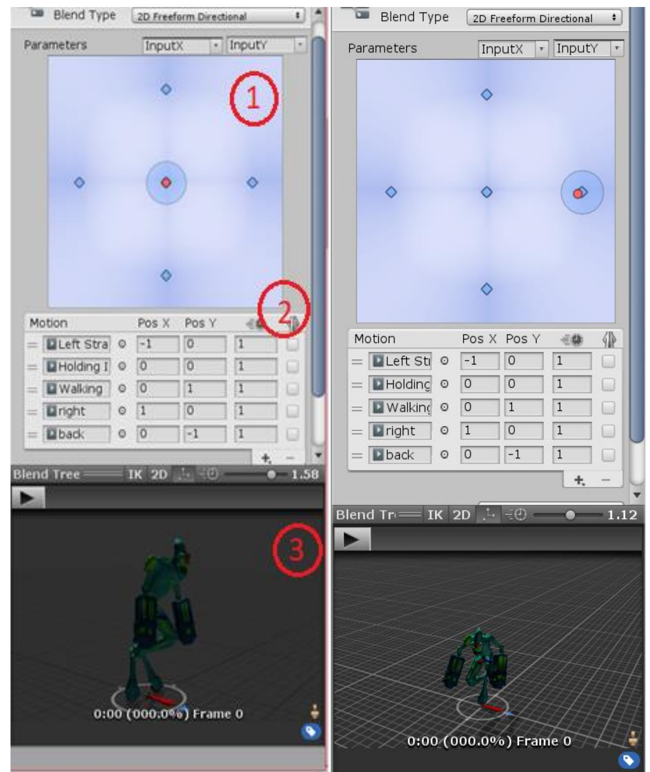



*Figura 8: Representacion de Blend Tree en Unity.*

## 4.5 Game Objects - RayCast: Disparos

Hasta ahora hemos logrado que AKONE, personaje principal de nuestro videojuego tipo *shooter* camine y salte.

En los videojuegos de tipo *shooter* es imprescindible disparar y para ello, utilizamos un componente de las físicas de Unity (*UnityEngyne.PhisicsModule*) denominado *RayCast*

La funcionalidad del *RayCast* se basa en la proyección de un vector que colisiona con cualquier objeto que se encuentre en su dirección y por cada objeto con el que haya colisionado, *RayCast* nos devuelve un booleano. El vector que se genera tiene 3 componentes: Posicion inicial (punto verde), modulo o rango (punto azul) y dirección (punto rojo) como se muestra en la figura 9.

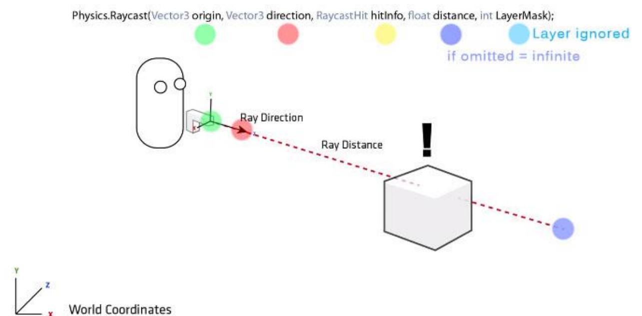



*Figura 9: Ejemplo de RayCast. Posición inicial (verde) dirección (rojo) y distancia o rango (azul).*

### 4.5.1 RayCast aplicado al disparo del Game Object AKONE

Para implementar el disparo en nuestro juego, lo primero que

vamos a comprobar es si el usuario ha generado un evento de disparo apretando la tecla izquierda del ratón. Para ello, se verifica el clic y con una variable "*RaycastHit*" se comprueba la posición inicial y final del *Ray* que genera el *RayCast.*

Si el usuario dispara, creamos una instancia de un objeto proyectil que nos da información de la posición de la bala, una variable dirección para orientar el proyectil y finalmente una acción de daño que tendrá la información del objeto con el cual ha colisionado el RayCast.

Cada vez que se genera un disparo, se crea un objeto proyectil dentro de nuestra escena del juego. Tres segundos después procedemos a destruirla con un evento *Destroy* para que no queden objetos innecesarios dentro de nuestra escena y así asegurarnos de que el rendimiento del juego no empeora.

Para la realización de pruebas y test de nuestro RayCast, vamos a crear un *script* de recibir daño el cual denominaremos *DamageTaking*. Aplicaremos este script tanto al personaje principal como a todo lo que consideremos que pueda ser destruido, en este caso los enemigos.

Este script tendrá una variable de tipo *int* que guardará la vida del objeto. En esta prueba nuestro objeto enemigo dispondrá de 3 puntos de vida. Por cada disparo efectuado, restaremos un punto de vida. Si el contador llega a 0, se destruye el objeto enemigo, tal y como se muestra en la figura 11.

En la parte superior de la figura 10 se muestra el vector representado por el *RayCast* (línea roja), atravesando a un enemigo (en este caso un prototipo de enemigo, una torreta roja). El *Ray* generado colisionará con él y lo atravesará, dándonos la información de que hay un objeto enemigo en la dirección del *RayCast*. En la parte inferior de la figura 10 se muestra el modo de ejecución que ve el usuario.

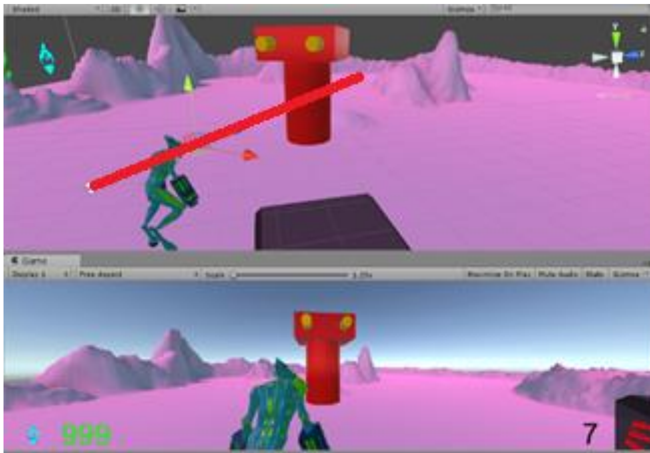



*Figura 10: RayCast atravesando a un enemigo.*

### 4.6 Game Object - Respawner

Cuando un enemigo muere, o si AKONE muere, puede volver a reaparecer (*respawn*) en el mapa. Para ello, se ha creado un script que contiene una variable llamada *delay* que representa el tiempo de reaparición del objeto en la escena.

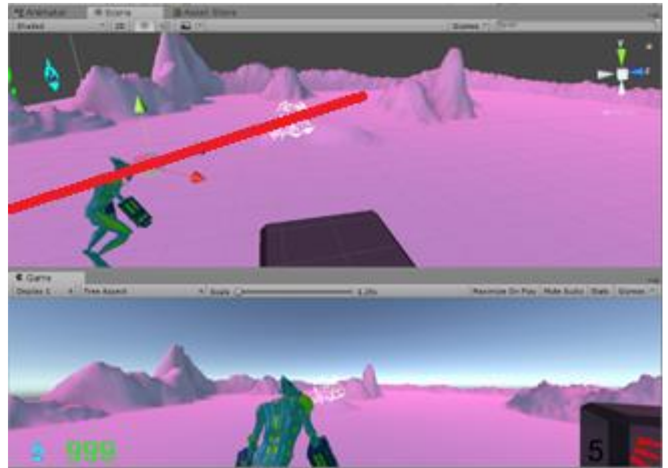



*Figura 11: Visualización del RayCast + collision con el objeto enemigo + evento Destroy.*

Otra variable *allRespawnPoints* representa cada uno de los puntos de respawn y una última variable *selectedPoint* es la selección del punto donde se produce la reaparición, obteniendo de forma aleatoria a partir de un vector de diferentes puntos de respawn situados en el mapa.

Los puntos de reaparición o respawns, los representamos en Unity mediante un Game Object vacío con una etiqueta *Respawn* y con fondo verde para identificarlos en nuestro mapa. Estos respawns no se verán físicamente a la hora de ejecutar el juego pero sí que los podremos ver en el modo de edición en Unity.

En la figura 12 se muestra un punto de *Respawn* que podremos situar libremente por el mapa gracias al componente *Transform* que nos permite desplazar el objeto, rotarlo y escalarlo.

*Figura 12: Respawn.*

### 4.7 Game Objects - Enemigos

Los enemigos siempre son importantes en un videojuego y presentan otro reto importante debido a las multiples formas que puede haber de implementarlos.

En un primer grado de desarrollo de un enemigo, se ha contemplado su movimiento y para ello, una IA básica.

#### 4.7.1 Inteligencia Artificial básica de los enemigos

Para mover un enemigo, Unity nos ofrece una herramienta denominada, las áreas de navegación (*Navigation Area*).

La opción *Navigator* nos ofrece un sistema de navegación que permite que algunos tipos de objetos (nuestros enemigos en este caso) puedan moverse de manera "inteligente" en diferentes áreas del juego. Para ello es necesario la creación de un área de navegación la cual podemos generar pulsando el botón *Bake*.

Estas áreas de navegación se pueden generar aplicando la opción de *Navigator*, situada a la derecha del *Inspector* tal y como se muestra en la figura 13.

La figura 14 muestra el mapa antes de aplicar el área de navegación y la figura 15 representa el mismo mapa con el area de navegación aplicada en el suelo. Unity nos representa el área de navegación como una malla azul turquesa encima del plano.

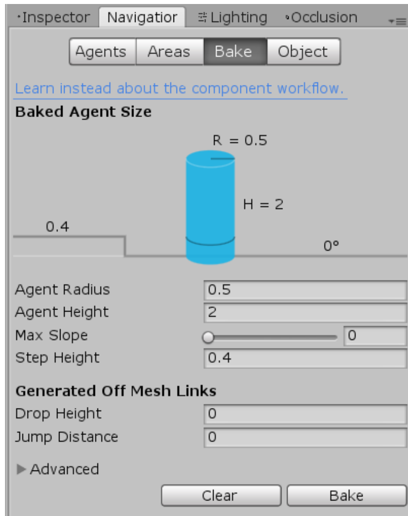



*Figura 13: Navigator de Unity.*

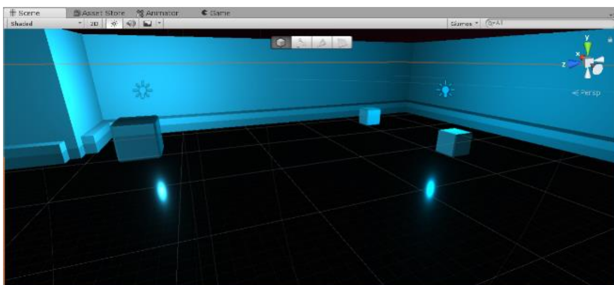


*Figura 14: Mapa del juego sin área de navegación.*

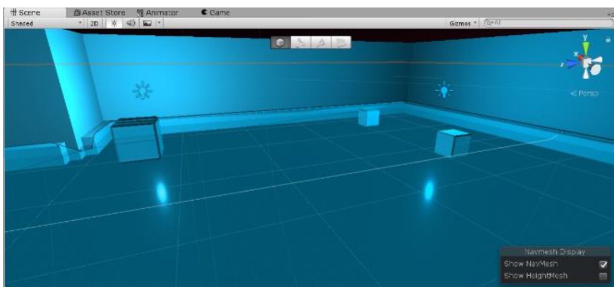


*Figura 15: Mapa del juego con área de navegación.*

Una vez representada el área de navegación, necesitaremos *Roaming Points*. Un *Roaming Point* es un punto en el plano que guia a nuestro enemigo por el área de navegación. Para ello, crearemos un objeto nuevo de Unity sin propiedades, vacío.

Este objeto lo vamos a etiquetar como punto de navegación y para diferenciarlo de un Respawn, utlizaremos un fondo azul (figura 16).

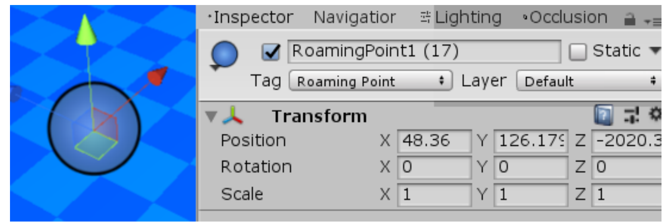



*Figura 16: Roaming Point.*

Hasta el momento hemos podido generar un área de navegación y puntos de navegación situados en diferentes partes del área de navegación.

Para mover los enemigos debemos realizar una conexión entre los puntos de navegación y un objeto que los identifique; un enemigo capaz de moverse de un punto de navegación a otro diferente de forma aleatoria.

Para lograrlo, lo primero que debemos hacer es asignar una nueva propiedad a nuestros enemigos, la propiedad *Nav Mesh Agent*.

Esta nueva propiedad (figura 17), es necesaria para que Unity identifique a nuestro enemigo como un objeto capaz de desplazarse por su área de navegación, dándonos la posibilidad de configurar diferentes valores:

*Nav Mesh Agent* nos ofrece la posibilidad de asignar qué tipo de:
·*Agent Type* será el tipo de objeto que queremos que se mueva por la malla generada, en este caso un personaje con el tag *"humanoide"*.
·*Base Offset* representa el desfase del cilindro de collision en relación al punto de pivote de *transform* (posición, movimiento y escala del objeto) permitiendo al enemigo colisionar a una cierta distancia, en este caso un valor de 1 metro.
·*Speed* siendo la velocidad máxima de movimiento que podemos asignar a nuestro enemigo.
·*Angular speed* o velocidad máxima de rotación con la que se girará el enemigo.
·*Acceleration*, la aceleración máxima.
·*Stopping distance* (el agente va a parar cuando esté cerca de la ubicación final.)
·*Auto Braking*, si se activa esta opción, el agente reducirá su velocidad cuando se acerque a su destino.

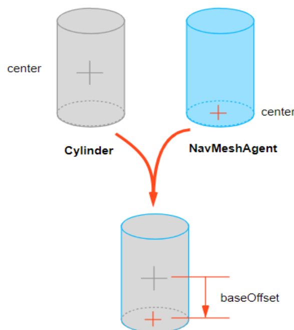
Además, los enemigos disponen de cuatro opciones para eludir obstáculos teniendo un *Radius* y *Height* asignado, un *Quality* como cantidad de obstáculos a esquivar (siendo *hight quality* nuestra opción asignada determinando que la cantidad de obstáculos a esquivar sea alta) y un *Priority* con un rango de valor de 0 a 99 indicando la prioridad de objetos que el agente ignorará al realizar la evasion donde un valor bajo para este atributo implica una prioridad más alta.

Disponemos de tres atributos más, el primero *Auto Traverse Off Mesh Link* que si se activa, el agente podrá atravesar

paredes invisibles (o enlaces *off-mesh*). *Con Auto Repath* activado, el agente intentará encontrar un nuevo camino parcial en caso de no tener un camino destino concreto. El camino parcial se genera hacia la ubicación más cercana al destino. Por ultimo, *Area Mask* nos permitirá decidir qué tipo de área debe considerar el agente a la hora de encontrar su camino. Esta área es la que se genera cuando pulsamos *bake* en la figura 13.

### 4.7.1 Detalles de Nav Mesh Agent.

El *Agent* está definido por un cilindro vertical cuyo tamaño está especificado por las propeidades *radius* y *height*. El cilindro se mueve con el objeto pero siempre permanece vertical incluso si se rota el objeto. La forma del cilindro es utilizada para detector y responder a colisiones entre otros agentes y obstáculos (figura 18).




*Figra 18: Funcionamiento de Nav Mesh Agent*

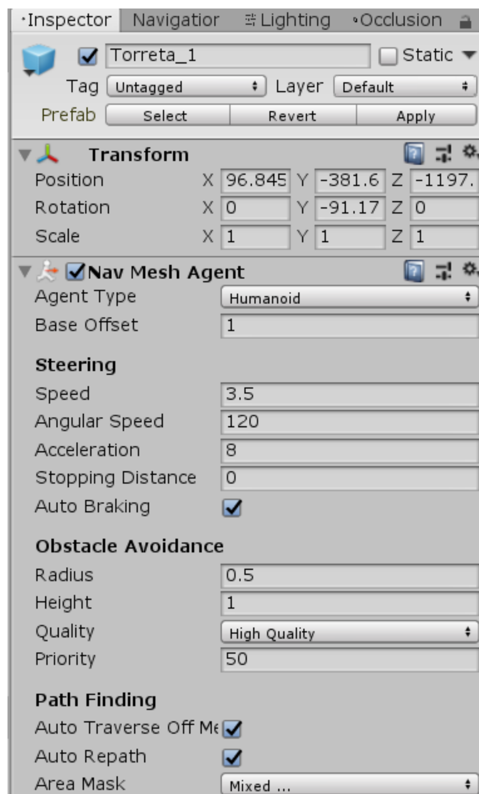



*Figura 17: Propiedad Nav Mesh Agent de Unity.*

### 3.5.2 Game Object - Enemy AI

*EnemyAI* es el script que se ha creado para la implementación del *Nav Mesh Agent* de nuestros enemigos. El script impulsa a nuestro objeto enemigo a moverse entre los diferentes puntos de posicionamiento en el área de navegación.

Su funcionamiento se basa en el comportamiento de diferentes estados en los cuales contemplamos si nuestro enemigo se encuentra en un área de navegación y si el enemigo ha detectado a nuestro *player* AKONE:

· Si el enemigo no se encuentra en un área de navegación, no podrá moverse.
· Si el enemigo se encuentra en un área de navegación, el enemigo se desplaza a un punto de navegación aleatorio.
· Si el enemigo se encuentra en un área de navegación y en su recorrido se cruza con el personaje principal, dispara.

Para ello contemplamos los siguientes estados del enemigo:

· *Roaming*: El enemigo se encuentra en un área de navegación y se desplaza libremente de un *Roaming Point* a otro.
· *RunRoaming*: El enemigo se encuentra dentro del área de navegación y cuando alcanza un *Roaming Point* se para durante un segundo, se gira rotando de 0 a 360º (dependiendo de dónde se encuentre el siguiente *Roaming Point*, girándose hacia la dirección del siguiente punto), y va al siguiente *Roaming Point*.
· *RunAttacking*: El enemigo se encuentra dentro de un área de navegación y en su trayectoria se cruza con nuestro *Player*. En este momento el enemigo se para por completo. Si el *Player* se encuentra dentro de un radio de 60º de cara al enemigo, el enemigo dispara. Si el *Player* sale de su radio de 60º, el enemigo vuelve al estado de *RunRoaming* para continuar con su trayecto. Este caso se aplica también cuando el enemigo mata al personaje principal, reanudando su trayecto hasta el siguiente *Roaming Point*.

La figura 19 muestra un esquema de cómo se mueve el enemigo por el área de navegación con el *Nav Mesh Agent* activado, al entrar en un *Navigation area* procede a moverse por el área de navegación de un *Roaming Point* a otro de forma aleatoria si solo si está dentro de un *Navigation Area*.

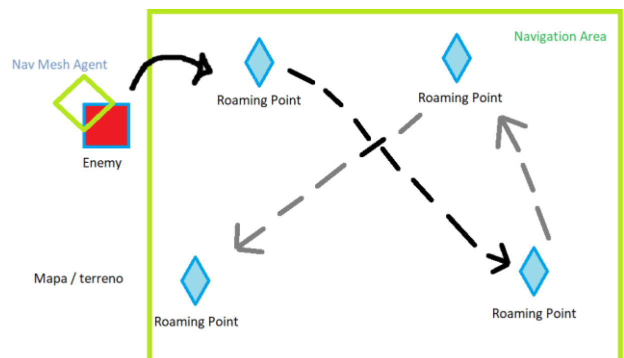



*Figura 19: Esquema del desplazamiento de un enemigo.*

### 4.8 Objetos

AKONE se compone de dos entidades: un alienigena y un humanoide, conformando así una dualidad de mente y cuerpo. Para que estas dos entidades puedan comunicarse y entenderse, el humanoide necesitará descifrar lo que su AKONE quiere y para ello la música será su canal de comunicación. Los Power Ups desenpeñan la función de conectar las dos entidades de AKONE y con ello, potenciar a AKONE con poderes ocultos.

#### 4.8.1 Game Objects - Power Ups

Los Power Ups se representan también como Game Objects de Unity. Para ello, crearemos un nuevo objeto al cual vamos a asignarle diferentes atributos con la finalidad de modificar el comportamiento de nuestro personaje principal AKONE.

Características de un Power UP:
-Irradiar luz de color llamativo (verde, amarillo, rojo, etc.)
-Su aspecto será en forma de cubo.
-Oscilará en el aire de abajo arriba y rotando en su eje 360º.
-Dependiendo de su color, un Power Up hará una cosa u otra.
· Verde: Cura al personaje principal.
· Rojo: Regenera la munición del personaje.
· Amarillo: Aumenta la velocidad de movimiento del personaje principal.
En la figura 20 podemos ver un Power Up de curación.

*Figura 20: Power Up de curación*

Un Power Up es un *Game Object* en forma de cubo con un componente de iluminación agregado (figura 21) el cual tiene un radio (rango) de 3 y una intensidad de 12 irradiando un color verde. Con la opción de *Draw Halo* activada se potencia la luminosidad mostrando una neblina del color de la luz.

#### 4.8.2 Scripts asociados a un Power Up

- *PickUp*: Este script interactúa con nuestro personaje principal. Cuando éste colisiona con él, el Power Up desaparece y reaparece nuevamente pasados unos segundos. Para que vuelva a aparecer, le agregamos una variable de tipo *int* de *Respawn* con un valor de 5.

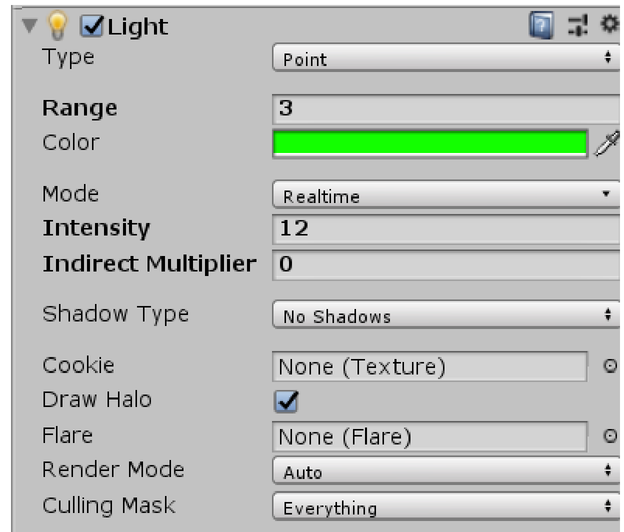



*Figura 21: Iluminación agregada al Power Up*

- *PickUp Bounce*: Este script es el encargado de darle movimiento a nuestro Power Up. Tiene 3 variables de tipo *float* que representarán el movimiento oscilante del objeto (arriba, abajo y rotación).
- *PicukUp_Health*: Hereda el comportamiento de *Damage Taking* llamando a la función de salud total del personaje principal para poder restablecer la vida de nuestro personaje principal al máximo en caso de haber recibido daño por un enemigo.
- *PickUp_Ammo*: Hereda el comportamiento del arma del personaje principal, regenerando por completo la munición del mismo.
- *PickUp_Speed:* Hereda el comportamineto del movimiento del personaje principal, llamando a la función de velocidad multiplicando su valor por 3 consiguiendo una velocidad el triple de rápida que nuestro personaje principal.

### 4.9 Game Object - Teletransportadores

Un teletransportador o *Teleporter* es un objeto conformado por dos puntos, inicio y final; los vamos a representar como un *Game Object* en forma de cuadrado (figura 22) que recibe la posición del personaje principal AKONE y la posiciona en otro lugar del juego llamado *TeleporPoint*.

Los *TeleportPoints* se representarán de forma similar a los *RespawnPoints*, pudiendo así desplazar cada punto de teletransporte en el lugar que consideremos logrando así teletransportar a nuestro personaje en la posición del escenario que deseemos.

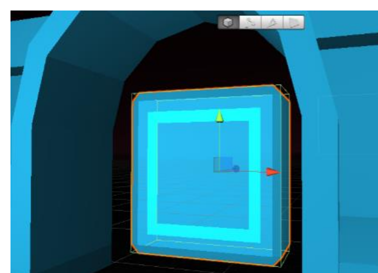


*Figura 22: Teletransportador*

### 4.10 Música y efectos de sonido

La música en nuestro videojuego es un factor muy importante y para ello, hemos creado un script para seleccionar y poder modificar los audios de manera que tengan opción de repetirse constantemente (*loop*) o modificar su volumen.

Creamos un *Game Object* vacío y le agregamos el script de *AudioManager*, que tendrá un *array* vacío para rellenarlo con las pistas de audio que importemos a nuestro proyecto (en formato *.mp3*). Nuestro script heredará las funciones de *AudioSource* (librería que reproduce audios en Unity) entre las cuales destacamos:

- *Volume*: Modificador del volumen del audio.
- *Pitch*: Modificador de la velocidad del audio.
- *Loop*: Modificador de la repetición del audio.

Y finalmente creamos una función llamada *Play* que nos permita reproducir nuestro audio del *array* que hemos ido llenando.

Si queremos que nuestro personaje active un sonido cuando coge un Power Up, añadimos una sentencia nueva en nuestro script de manejo de Power Ups que llama al *AudioManager* y le especificamos con una cadena de caracteres el nombre del audio que tiene que reproducir.

## 5. Resultados

Se han realizado diferentes tipos de test para minimizar los errores a nivel de *software* desarrollado, de los que destacamos:

- *Test unitario de Unity:* con la herramienta *Unity Test Tools* permitiendo corroborar en algunas clases su funcionamiento ideal.
- *Tests de rendimiento:* sobrecargando el rendimiento de Unity generando un número elevado de objetos permitiendo así saber hasta dónde es capaz de aguantar nuestro juego sin ralentizarse (figura 24)
- *Tests de caja negra:* Se ha mostrado el juego finalizado a diferentes personas ofreciéndoles a cambio realizar una pequeña encuesta en la cual se les preguntaba entre otras cosas qué tipos de fallos habían detectado.

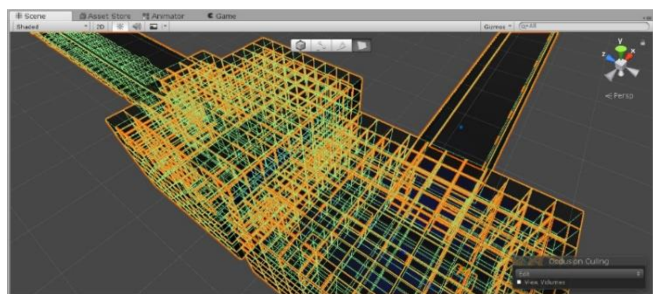


*Figura 24: Test de rendimiento sobrecargando el número de objetos que debemos tener en la escena; esta prueba contenía 1000 cubos con propiedades de Mesh Collider.*

## 6. Conclusiones y mejoras

Se ha diseñado, implementado y testeado un videojuego jugable con inicio, cinemáticas, interfaz de usuario accesible en cualquier momento del juego y un nivel dividido en 3 partes.

Hemos realizado dos documentos previos a la implementación siguiendo la mecánica que se utiliza en la indústria:

- Documento de diseño.
- Documento de implementación.

Como conclusiones podemos decir que los resultados obtenidos al final del proyecto han sido satisfactorios.

A título personal he aprendido el uso de nuevas herramientas para el desarrollo y aprendizaje de como crear un videojuego desde cero sin tener nociones previas de ello.

Otro aspecto a tener en cuenta ha sido el hecho de poder realizar el proyecto de forma individual ya que he tenido la oportunidad de organizarme y llevar el proyecto con detalle de forma que he podido aprender a administrar el tiempo a la hora de marcarme objetivos y metas.

En cuanto a mejoras en el videojuego destacamos las siguientes:

- Corregir animaciones del personaje principal y añadir animaciones a los enemigos para conseguir un juego más inmersivo.
- Desarrollo de un *Sistema Online* permitiendo jugar a varias personas a la vez para entender el funcionamiento de las redes aplicadas a los videojuegos.
- Crear un modo história del juego, añadiendo diferentes niveles de dificultad (facil, normal, dificil), seguido de un sistema de trofeos y recompensas para incentivar a jugadores nuevos y con ello conseguir una duración de las partidas mayor.
- Añadir diferentes armas, Power Ups y niveles para tener un juego mucho más completo.
- Creación de una plataforma *Patreon* con el objetivo de llegar a poder publicar el videojuego en *Steam*.
- Desarrollar una version del juego para *Android* e *IOS.*

## 7. Agradecimientos

En primer lugar me gustaría agradecer a mis padres y mi hermano todo el apoyo recibido en el transcurso del proyecto.

Agradecer infinitamente la oportunidad que he tenido de conocer a mi tutor Enric Martí Gòdia ya que el ha sido el pilar fundamental de todo mi proyecto y guia hacia un futuro posible.

También agradecer a mis amigos Víctor Escobedo, Sergio Paulo Rodríguez, Adrián Bueno, Rubén Serrano, Toni Villar y Óscar López que hayan podido estar a mi lado apoyándome y animándome a seguir adelante con el proyecto.

Y por ultimo agradecer el tiempo dedicado a todos los que se han molestado en probar AKONE y dárme su opinion al respecto.

## 8. Biliografía y referencias.

[Animation and riding]
https://assetstore.unity.com/packages/tools/animation/puppet3d-111554 (Fecha de último acceso Enero 2019)

[Animaciones]
https://www.mixamo.com/ (Fecha de último acceso Octubre 2018)
https://www.autodesk.es/products/3ds-max/overview (Fecha de último acceso Octubre 2018)

[Ammo Pick Up]
https://www.youtube.com/watch?v=OtivTjh_Oc8 (Fecha de último acceso Enero 2019)

[Countdown Timer]
https://www.youtube.com/watch?v=E6qEPJUAZNk&t=468s (Fecha de último acceso Diciembre 2018)

[Comparativa Unity Vs Unreal Engine]
http://www.hagamosvideojuegos.com/2015/03/unreal-engine-4-vs-unity3d-5.html (Fecha de último acceso octubre 2018)

[Diseño y Texturizado]
https://www.adobe.com/Photoshop/ (Fecha de último acceso Octubre 2018)

[Lynda] – Unity User Interfaces and Animation
https://www.lynda.com/Unity-tutorials/Cert-Prep-Unity-User-Interfaces-Animation/536420-2.html?srchtrk=index%3a13%0alinktypeid%3a2%0aq%3aui+unity%0apage%3a1%0as%3arelevance%0asa%3atrue%0aproducttypeid%3a2 (Fecha de último acceso Diciembre 2018)

[Modelaje 3D]
https://www.autodesk.es/Store_3ds-Max (Fecha de último acceso Octubre 2018)

[Mallado]
https://uvlayaout.com/ (Fecha de último acceso Octubre 2018)

[Udemy] - Desarrollo de un videojuego 2D-3D
http://www.udemy.com/complete-csharp-unity-2d-3d-game-development-masterclass-2018/ (Fecha de último acceso Noviembre 2018)

[Udemy] – Unity Tech Art: Realistic Lighting For Games Development
https://www.udemy.com/lighting-in-unity/ (Fecha de último acceso Diciembre 2018)

[Unity]
https://unity3d.com/es (Fecha de último acceso Enero 2019)

[Unity] – Game Objects
https://docs.unity3d.com/es/current/Manual/GameObjects.html (Fecha de ultimo acceso Noviembre 2018)

[Unity] – Nav Mes Agent
https://docs.unity3d.com/es/current/Manual/class-NavMeshAgent.html (Fecha de último acceso Diciembre 2018)

[Unity] - Sripting
https://unity3d.com/es/learn/tutorials/s/scripting (Fecha de último acceso Diciembre 2018)

[Unity] - Colliders
https://docs.unity3d.com/es/current/Manual/CollidersOverview.html (Noviembre 2018)

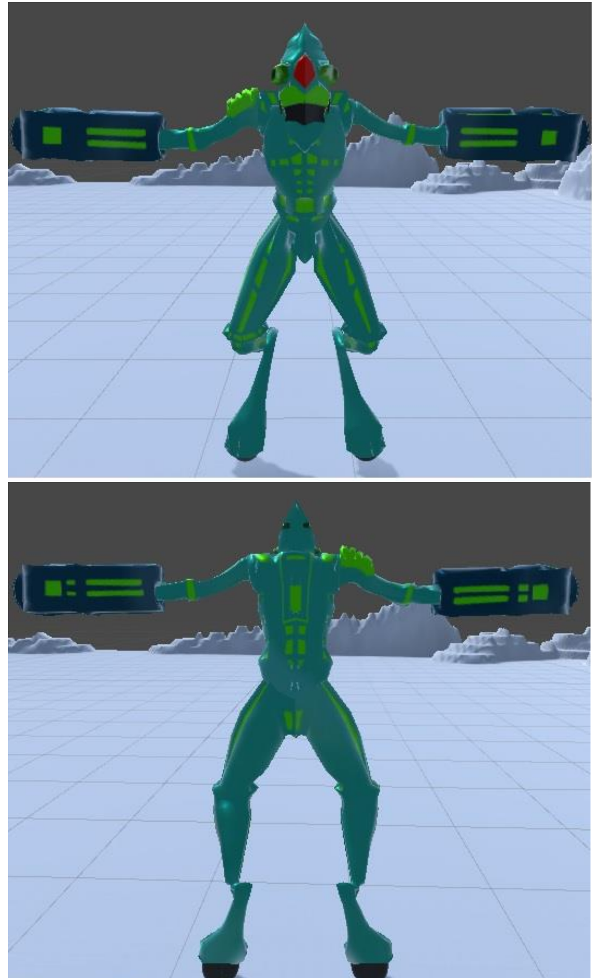
[Unity] - RayCast
https://docs.unity3d.com/2018.2/Documentation/Manual/Raycasters.html (Fecha de ultimo acceso Diciembre 2018)

[Unity] – Navigation areas
https://docs.unity3d.com/Manual/Navigation.html (Fecha de ultimo acceso Diciembre 2018)

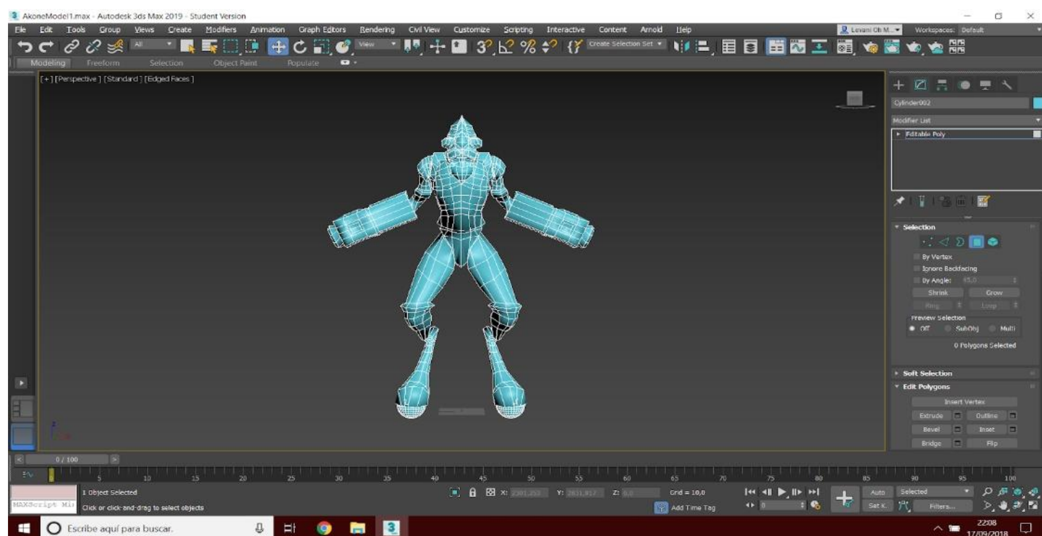
[ThirdPersonController]
https://www.youtube.com/watch?v=u_zn5zMjirc&index=4&list=PLJfktYG5YLJGK-ROk1Gj1_i8AtnIyeKTs (Fecha de último acceso Octubre 2018)

## Anexo 1. Boceto y diseños de AKONE

*Boceto de AKONE dibujado el día (25/07/2016)*



*Primera implementación de AKONE en Unity*




*Modelaje de AKONE en 3DsMax Studio*

**Anexo 2. Diagrama de clases del juego AKONE**

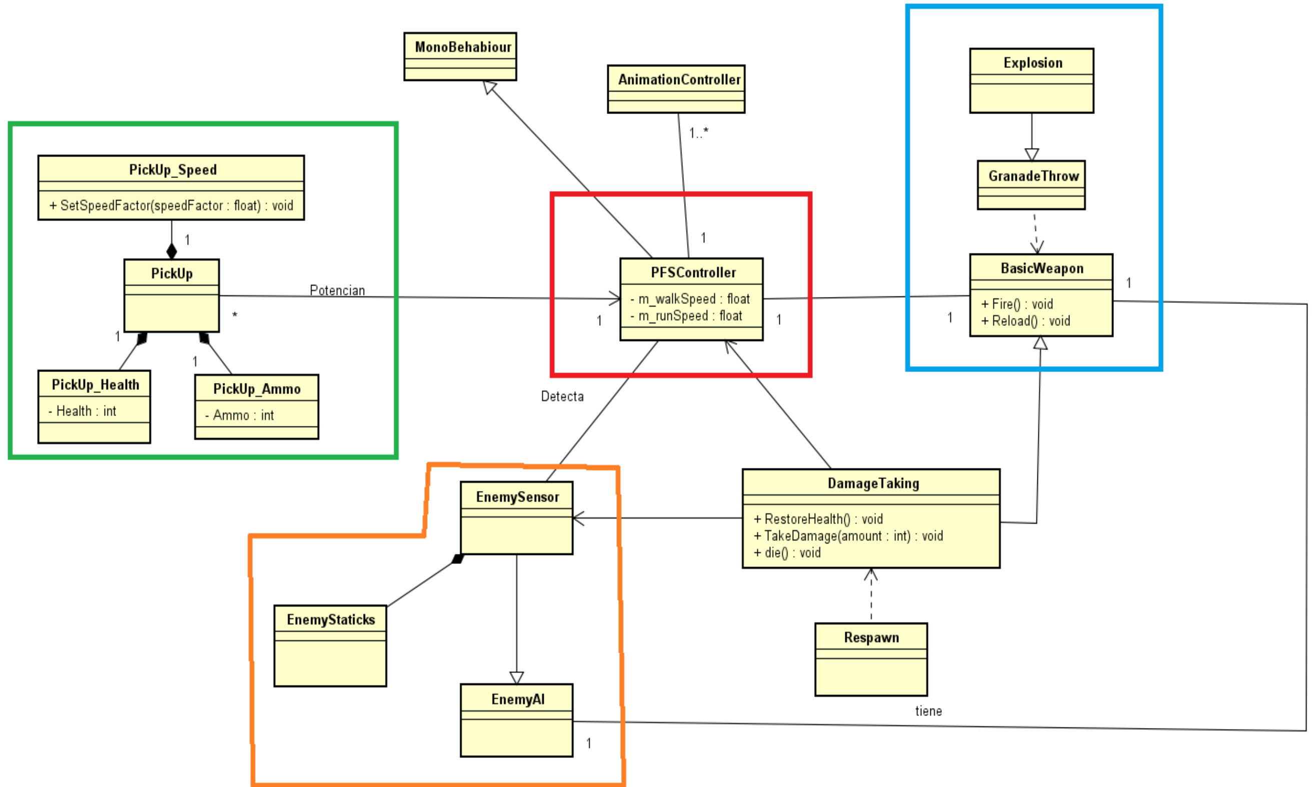



Diagrama de clases AKONE:

1. Todo el proyecto hereda de la clase **MonoBehabiour** (clase propia de Unity que contiene toda la interacción con físicas y métodos para la interpretación de todo lo que son GameObjects de Unity y su comportamiento).
2. **FPSController**: Clase principal del proyecto (marcada em rojo); Contiene todo lo que sería el movimiento del personaje (forward, back, left, right) y el salto, con propiedades como velocidad y gravedad, el control de la cámara y la lectura de inptus de entrada por teclado (A , W , S , D , barra espaciadora).
3. **BasicWeapon**: Clase que representa las armas del juego (marcada en azul), objeto capacitado de generar un proyectil y destruirlo; **GranadeThrow** hereda el comportamiento de **BasicWeapon** y contiene las físicas necesarias para generar una parábola y un tiempo de recorrido del proyectil generado. A su vez, **Explosion** hereda de **GranadeThrow** permitiendo al proyectil generar físicas a la hora de colisionar con otros objetos añadiéndoles una fuerza y dirección seguida de un temporizador para detonar el proyectil generado.
4. **DamageTaking**: Clase que permite modificar la salud de los objetos necesariamente dañados por los proyectiles que genera **BasicWeapon**. Además, contiene un método para destruir objetos.
5. **Respawn**: Clase que permite la creación de la instancia del personaje o de los enemigos tras haber muerto en una posición X, Y, Z (hereda de **DamageTaking**)
6. **EnemyAI**: proporcionando la inteligencia artificial básica del enemigo (marcada en naranja) la cual permite el cambio de estados de **Enemy Staticks** decidiendo cuándo puede o no puede moverse y cuando puede disparar o no puede).
7. **EnemyStaticks**: Clase heredada de **EnemyAI** que maneja el estado del enemigo (entendemos por estado el posicionamiento del enemigo; si está quieto, si se mueve o se si mueve atacando).
8. **EnemySensor**: Clase que hereda de **EnemyAI** que detecta si hay un objeto Player en un radio definido por unas constantes de posición y dirección. Esta clase se encarga de hacer que un enemigo dispare, pero para ello necesita tener las propiedades de **BasicWeapon**.
9. **PickUp:** Clase que permite al Player modificar sus atributos (marcada en verde); De esta clase dependen las clases **PickUp_Speed** (modificando el comportamiento de velocidad del player) **PickUp_Health** (modificando la salud del player) y **PickUp_Ammo** (modificando la munición de **BasickWeapon**).

- Un player solo puede tener un **basicWeapon**; un enemigo solo puede tener un **basicWeapon.**

**Anexo 3: imágenes del juego AKONE finalizado.**

Interfaz de usuario:

- En la parte superior izquierda se observa un icono que representa la vida de AKONE y un valor numérico siendo este la cantidad de disparos que puede recibir el personaje con un valor máximo de 3.
- En la parte superior derecha se observan 2 iconos que representan la munición del personaje principal con un valor máximo de 12 y las granadas con un valor máximo de 2.
- En la parte superior central se observa un temporizador decrementativo, si llega a 0, el juego se pausa automáticamente generando un menu de usuario que permite reiniciar el nivel o salir de la aplicación.
- En la parte inferior derecha se puede apreciar la cantidad de puntos del personaje (*score*) pudiendo ser un valor positive o negative dependiendo del número de enemigos destruidos acumulando 25 puntos por destruir MegaAKONEs_Gamma(naranjas) o 50 puntos por destruir MegaAKONEs_Aqua (azules). En caso de morir, al personaje principal se le restan 100 puntos del valor acumulado.
- En la parte inferior derecha se observa también una cámara enfocando hacia la parte trasera del personaje pudiendo así tener un control en todo momento de lo que sucede detrás de nuestro personaje.
- En el transcurso del juego, aparecerán mensajes (*Akonsejos*) situados a la izquierda de la pantalla que informarán al usuario de cómo interactuar con su personaje y anticiparle de de algunos acontecimientos (*Akontecimientos*).