



COLEGIO LA INMACULADA PADRES ESCOLAPIOS

Proyecto Tiger - 1



RAÚL ALMOROX GUERRERO Y MARIO NAVARRETE CARRO

Índice

1. Resumen	3
2. Introducción	6
3. Tecnologías	7
3.1 Unity	7
3.1.1 ¿Qué es Unity?	7
3.1.2 Requisitos	9
3.1.3 Instalación	9
3.2 Asset Store	12
3.2.1 ¿Cómo funciona?	12
3.2.2 Assets	15
3.3 Vuforia	20
3.3.1 ¿Qué es Vuforia?	20
3.3.2 ¿Cómo instalar Vuforia?	21
3.4 Arduino	27
3.4.1 Configurar	30
4. Objetivos	49
5. Desarrollo	50
6. Conclusiones	96
7. Dificultades	97
8. Bibliografía	98

1. Resumen

Nuestro proyecto trata de desarrollar un videojuego de realidad aumentada en el que se utiliza una maqueta de un tanque con una cámara instalada para enviar la imagen a un ordenador. La imagen obtenida será utilizada como escenario del videojuego.

Respecto al videojuego, será un shooter (videojuego de disparos) en primera persona en el que tendremos que destruir una serie de objetivos y obtener la mayor puntuación posible. Este será ejecutado sobre un ordenador portátil bajo el motor de videojuegos [Unity Engine](#).

La maqueta, será un robot basado en el tanque Tiger-1 de la segunda guerra mundial, cuyo funcionamiento se basa en [Arduino](#), este utilizará diferentes motores, para el movimiento típico de un tanque (moverse hacia delante, girar la torreta superior y subir y bajar el cañón), además de diferentes elementos de comunicación.

Para la comunicación entre el ordenador y la maqueta hemos optado por varias soluciones. Para la emisión de video hemos utilizado un emisor y un receptor de radio similar a como se utiliza en los drones FPV. Respecto a la comunicación entre Unity y Arduino hemos combinado el uso de la librería de Unity, Uduino, y una placa de wifi para Arduino.

Este proyecto nace a partir de la salida al mercado de [Mario Kart Home Circuit](#) por parte de la compañía de videojuegos Nintendo.

Por último, debemos destacar que la finalidad del proyecto es la investigación, ya que nos hemos centrado en aprender sobre diferentes tecnologías y a la integración de estas, aunque, a priori, no tengan mucha relación entre sí.

1. Resume

Our project aims to develop an augmented reality video game using a model of a tank with a camera installed to send the image to a computer. The image obtained will be used as a stage for the game.

Regarding the video game, it will be a shooter (shooting videogames) in first person in which we will have to destroy a set of targets and obtain the highest possible score. It will be executed on a laptop under the videogame engine Unity Engine.

The model, will be a robot based on the tiger 1 tank of the second world war, whose operation is based on Arduino, it will use different engines, for the typical movement of a tank (move forward, rotate the upper turret, raise and descend the cannon), in addition to different elements of communication.

For the communication between the computer and the model we have opted for several solutions. For video transmission we have used a radio transmitter and receiver similar to those used in drones FPV. For the communication between Unity and Arduino we have combined the use of the Unity library, Uduino, and a wifi board for Arduino.

This project was born out of the launch of Mario Kart Home Circuit by the videogames company Nintendo.

Finally, it should be mentioned that the purpose of the project is the investigation, because we have focused on learning about different technologies and the integration of these technologies, although, a priori do not have much relationship with each other.

2. Introducción al proyecto

El proyecto surge del inicio de un proyecto personal cuyo fin era aprender a usar Arduino mediante la creación de un pequeño tanque con chasis de madera, al que se le añadiría un par de motores y un receptor Bluetooth, aunque nunca llegó a ser completado.

Cuando Nintendo presento su videojuego, se nos ocurrió aprovechar la idea del tanque, y juntarlo con Unity para hacer algo similar al videojuego de Nintendo.



Al principio había muchas dudas sobre la viabilidad del trabajo puesto que había varias cuestiones que había que resolver. La primera era la obtención de la imagen a partir de una cámara instalada en el tanque. Tras una exhaustiva investigación decidimos usar un sistema similar al que se utiliza en los drones FPV, Una cámara simple y un emisor de radio a 5.8Ghz, además de un receptor que envía la imagen por UVC, el estándar que usan las webcams.

De esta manera con la librería de Unity, Vuforia, podríamos acceder fácilmente al video de la cámara.

Otra de las cuestiones que tuvimos que resolver fue la conexión entre Unity y Arduino. En un principio comenzamos el desarrollo de un sistema para comunicarlos con Bluetooth, este desarrollo fue rápidamente desechado, ya que el modulo Bluetooth era demasiado lento, tenía muy poco alcance y había mucha inestabilidad en la conexión.

Más tarde descubrimos una librería en Unity que nos permitía conectar ambos sistemas mediante el uso de un receptor wifi. De esta forma no sólo la conexión es más estable si no que es mucho más rápida y con mucho más alcance.

Respecto al videojuego, hemos utilizado diferentes assets de la Asset Store de Unity para obtener diferentes modelos de tanques y aviones de la segunda guerra mundial.

Al no tener disponible un giroscopio y un acelerómetro (elementos que habitualmente se usan en la realidad aumentada para capturar el movimiento), nos vimos obligados a sincronizar el movimiento de la maqueta con el movimiento dentro de Unity mediante código, intentando imitar el comportamiento de la maqueta.

3. Tecnologías Utilizadas

En este apartado vamos a ir desarrollando primeramente los programas y tecnologías usadas, un breve resumen de que tratan, que desempeño general tienen y en que hemos usado nosotros estas herramientas para poder así hacer nuestro proyecto

3.1 Unity

3.1.1 ¿Qué es Unity?

Unity es un [motor de videojuegos](#) multiplataforma creado por Unity Technologies es una plataforma que permite el desarrollo desde Windows, Linux, MacOS, además, aparte permite la creación y compilación de videojuegos desde diferentes dispositivos o plataformas como Android o iOS.



Unity Technologies fue fundada en 1988 por David Helgason, Nicholas Francis, y Joachim Ante en Copenhagen. Tras su primer fracaso y sabiendo el poder del motor de desarrollo que tenían entre manos, se propusieron remodelarlo de tal manera que fuese accesible para todos los programadores incluidos amateurs/freelance en sus motores 2D y 3D. En el año 2008, ya contaba con una base de 1 millón de desarrolladores.

El boom de Unity comienza en 2005 en la conferencia mundial de desarrolladores de Apple, donde se lanzó su primera versión, aunque solo funcionaba sobre la plataforma Mac.

En 2010 se lanzó Unity intentando buscar así el interés de las grandes empresas, aparte de seguir mejorando las herramientas proporcionadas a los desarrolladores independientes.

Su última versión, [Unity 5](#), se anunció en 2015 en la conferencia Game Developers, esta nueva versión incluye añadidos como Mecanim Animation, soporte para [DirectX 11](#) y soporte para juegos en Linux y arreglo de bugs y texturas.

Esta plataforma ofrece la posibilidad de poder [probar tu juego](#) mientras es desarrollado, haciendo así que la experiencia de programar sea más visual sabiendo en qué está fallando tu proyecto en todo momento.

Todo Unity se mueve a través de [Scripts](#) para darle utilidad a los objetos creados por el usuario, estos se mueven en [C#](#) como lenguaje de programación, lo cual nos supuso un pequeño desafío a la hora de crear nuestro proyecto.

Aunque también hay que elogiar la facilidad con la que un usuario principiante, como puede ser nuestro caso. Al crear el proyecto puede acceder a la tienda de Unity llamada "[Asset Store](#)" y descargar un extenso catálogo de assets para el proyecto de una manera sencilla, facilitando así el trabajo de los pequeños programadores que puede que no tengan los recursos necesarios para modelar diseños, texturas, etc...

3.1.2 Requisitos de Unity

Unity tiene unos **requisitos muy bajos** para poder ser instalado, aunque a la hora de programar siempre se agradecerá tener cuanto mejor el ordenador para poder tener acceder a todas sus funcionalidades de una forma más fluida.

Requisitos:

- Windows 7 SP1+, 8, 10. Solo la versión de 64 bits es compatible.
- Mac OS X 10.12+
- Ubuntu 16.04, 18.04, and CentOS 7.
- **GPU:** Tarjeta de video con capacidad para DX10 (shader model 4.0).

3.1.3 Instalación de Unity

Desde la página oficial de **Unity Engine** lo primero que vemos es un apartado para descargar, una vez dentro debemos elegir la versión de Unity que queramos, nos encontraremos con que hay diferentes planes gratuitos, aunque debemos cumplir una serie de requisitos.

Descargar Unity

¡Bienvenido! Está aquí porque desea descargar Unity, la plataforma de desarrollo más popular del mundo para crear juegos multiplataforma y experiencias interactivas 2D y 3D.

Antes de descargar, elija la versión de Unity que sea adecuada para usted.

[Elige tu Unity + descargar](#)[Descarga Unity Hub](#)

[Descubrir más acerca del nuevo Unity Hub aquí.](#)

Descargar Unity Beta

Obtén acceso antes que los demás a nuestras funciones más recientes, y ayúdanos a mejorar la calidad haciéndonos conocer tus valiosos comentarios.

[Descargar versión beta](#)

¿Ya eres cliente?

Excelente. Descarga Unity aquí si tienes una suscripción Plus o Pro.

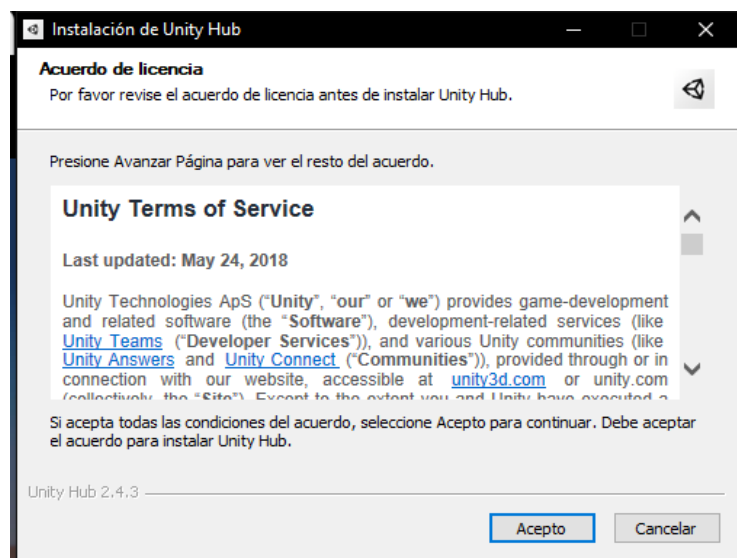
[Descargar](#)

Echando un vistazo rápido estos son los planes estándar para empresas dentro de Unity, pero nosotros al ser Estudiantes tenemos una opción especial, igualmente si quieres [descargarlo gratuitamente](#) como individuo también tienes tu opción.

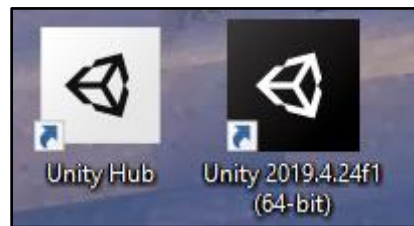
El único requisito para esta licencia gratuita es que no generes más de 100.000 dólares mediante juegos creados en su plataforma al año.

Plus	Pro	Empresa
Más funcionalidad y recursos para potenciar tus proyectos	Soluciones completas para que los profesionales creen y operen	Éxito a gran escala para organizaciones grandes con objetivos ambiciosos
399 \$ /año por puesto <small>Plan anual, prepago anual</small>	1.800 \$ /año por puesto <small>Plan anual, prepago anual</small>	2.000 \$ /mes por 10 puestos <small>Plan anual, pagado mensualmente</small>
Elegir el plan Conoce más	Elegir el plan Conoce más	Elegir el plan Conoce más
<ul style="list-style-type: none">✓ Versión más reciente de la plataforma de Unity✓ Personalización de la pantalla de inicio✓ Análisis de operaciones en vivo✓ Diagnóstico en la nube en tiempo real	<ul style="list-style-type: none">✓ Todo en Plus✓ Hasta tres licencias personales de Unity Teams Advanced✓ Prioridad en el servicio al cliente✓ Acceso prioritario a Success Advisors+ Opciones personalizadas disponibles<ul style="list-style-type: none">+ Soporte técnico+ Servicios para el éxito integrados+ Acceso al código fuente+ Capacidad de licencias de Build Server	<ul style="list-style-type: none">✓ Todo en Pro✓ Soporte técnico✓ Capacidad de licencias de Build Server✓ Administrador del éxito del cliente✓ Plan de aprendizaje personalizado✓ Sesiones de Learn Live de Enterprise (4) <div><small>Para obtener información sobre las opciones personalizadas de Enterprise, incluidos los servicios para el éxito integrados, licencias avanzadas, acceso al código fuente, soluciones del sector, más de 150 puestos o facturación, comuníquese con Ventas de Unity.</small></div>

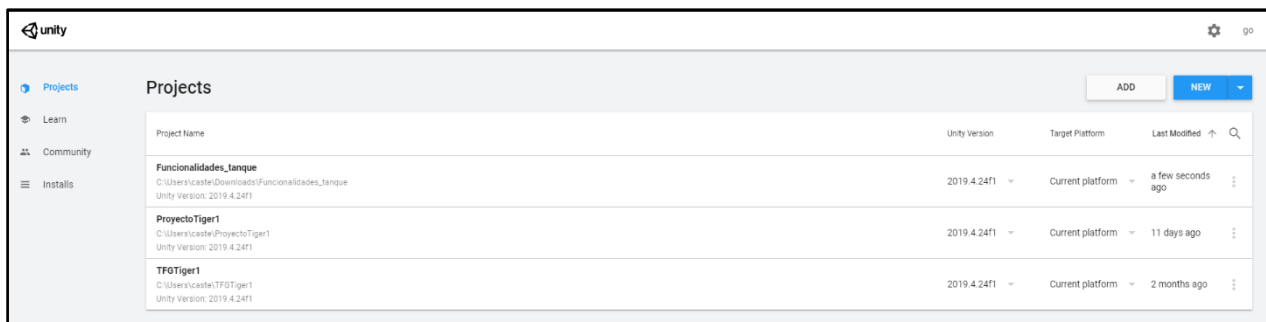
Simplemente daremos a descargar el instalador, una vez descargado daremos a siguiente para que se complete la descarga aceptando sus términos y condiciones.



Ya con la plataforma descargada nos encontraremos con dos accesos directos a los que ya podemos acceder.



Unity Hub es desde donde se accede a todos nuestros proyectos y la versión de Unity sobre la que trabajaremos, que es la última versión de normal, en este caso existen versiones anteriores, pero nosotros hemos usado la **versión LTS de Unity 2019**, que ofrece más estabilidad y mejor compatibilidad con algunas de las bibliotecas que veremos más adelante.

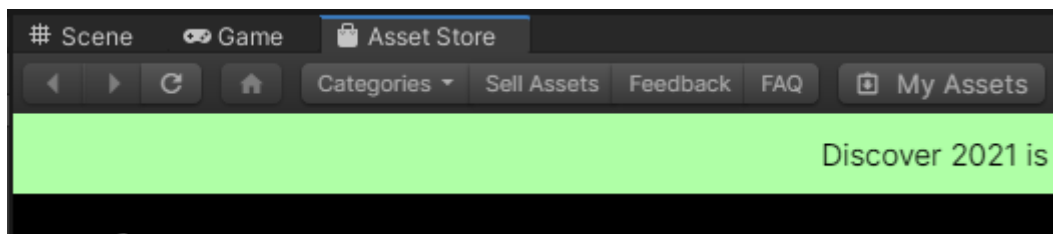


Ya dentro del Hub tenemos nuestros diferentes proyectos, la opción de abrir un nuevo proyecto o crear uno nuevo eligiendo así el proyecto que queremos hacer.

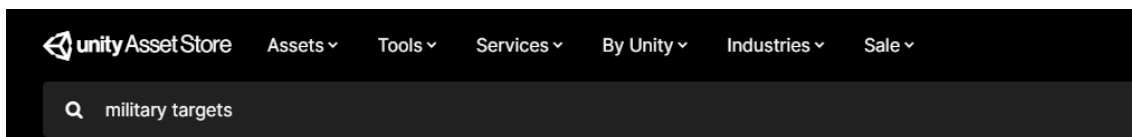
3.2 Asset Store

3.2.1 ¿Cómo funciona?

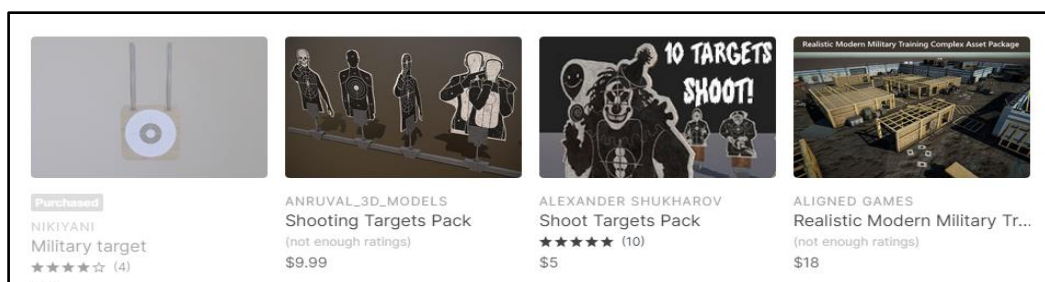
La asset store es una [tienda](#) implementada dentro de Unity que te permite acceder a ella de una manera muy cómoda con las ventanas principales.



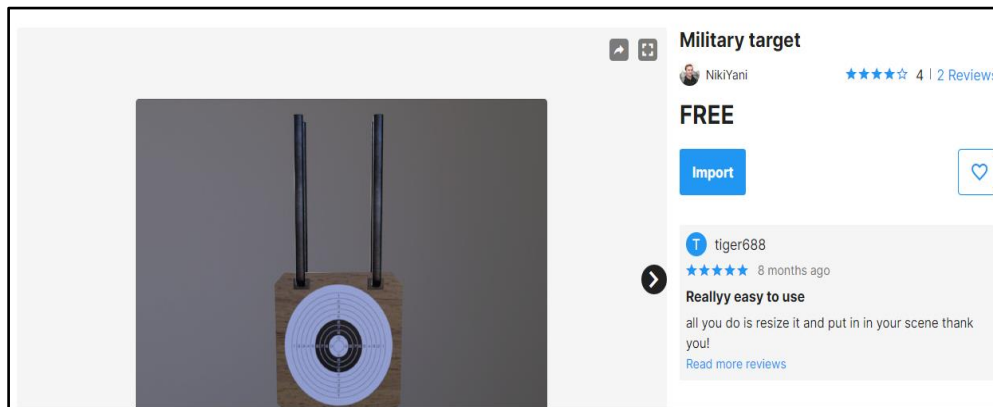
En este caso vamos a seguir el proceso para ver el funcionamiento de la Asset Store de cómo nosotros implementamos nuestras dianas en el juego.



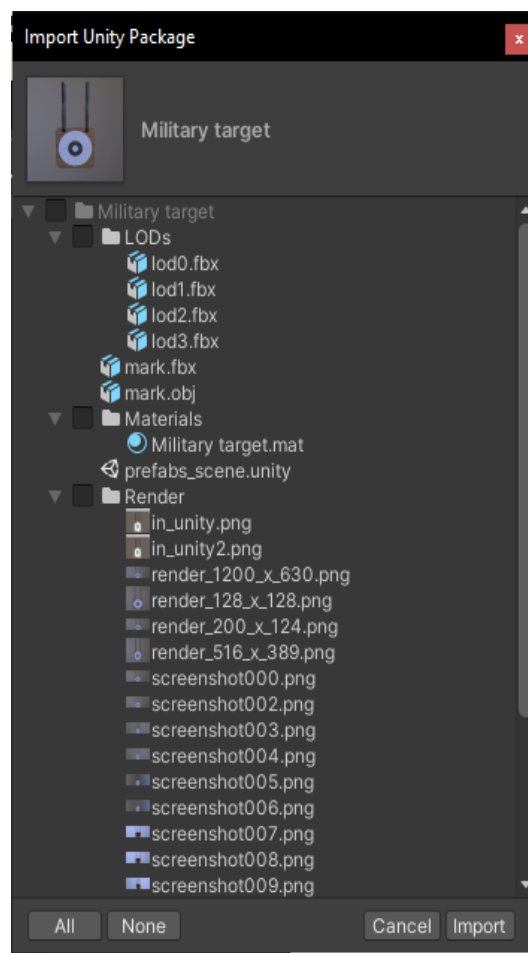
En la [barra de búsqueda](#) pones lo que te interesa encontrar, encontraremos diferentes desde gratis a con precio en este caso nosotros elegiremos la diana gratuita que es la que hemos usado que aparte se ve muy bien.



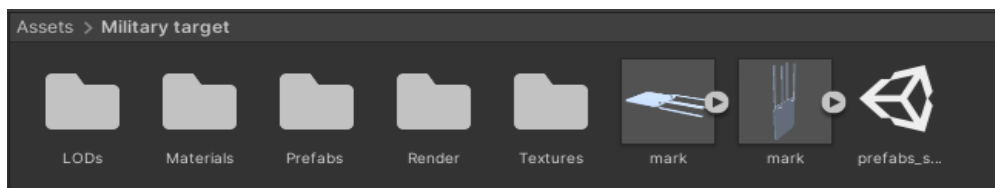
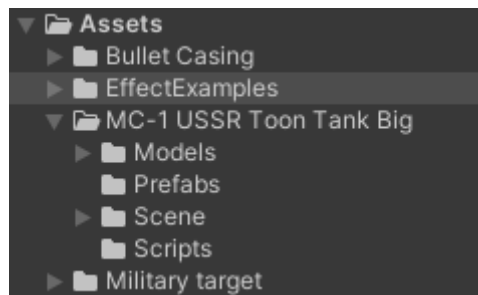
Dentro de la página tendrás una pequeña **preview** de lo que vas a descargar.



Simplemente tendrás que dar a descargar y más adelante a “**import**”. Al dar a importar te salta esta ventana:



Con todos los componentes que contiene nuestra diana, simplemente das a importar y te vas en tu proyecto al apartado de Assets:

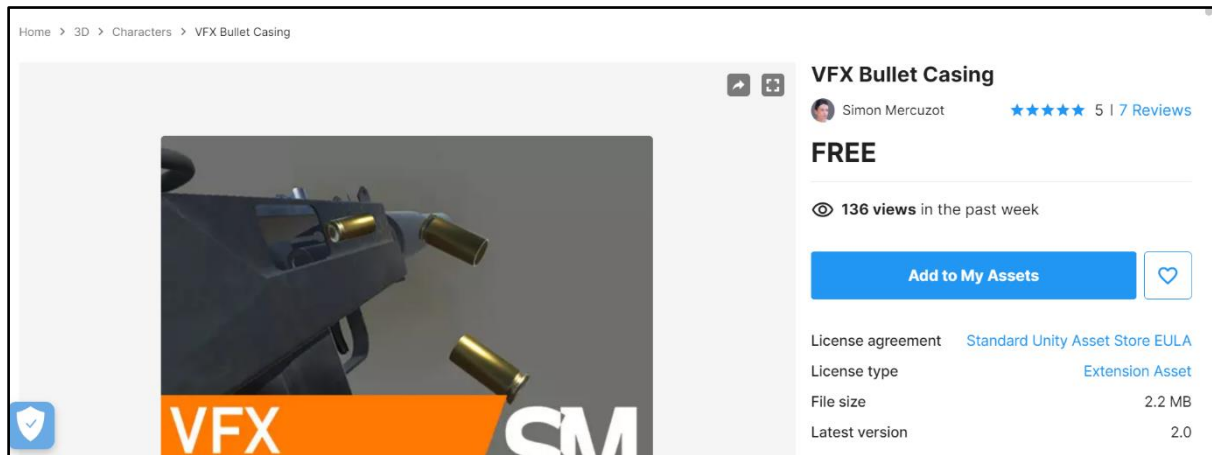


Ya dentro de la vista tenemos toda la diana importada simplemente si la queremos usar podemos o arrastrar el [Prefabs](#) de la diana o crear un objeto e implementarse este asset.

Esta tienda nos ha sido muy útil a lo largo del proyecto y de aquí salen todos nuestros objetivos, desde los tanques, aviones y dianas junto con las explosiones.

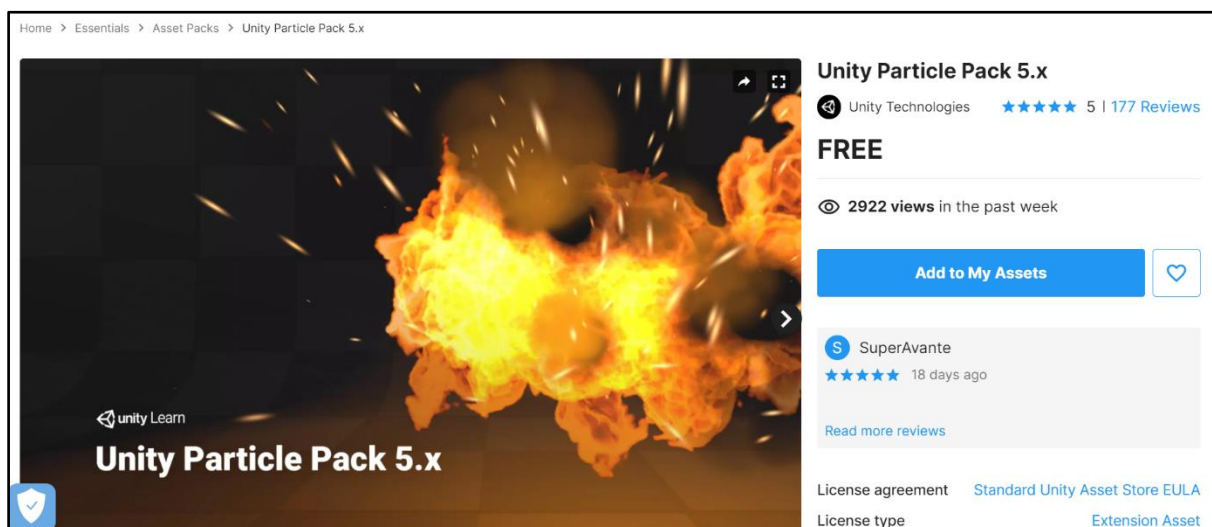
3.2.2 Assets utilizados

- *Example_Gun:*



Nuestro primer asset que vamos a comentar es example_gun de VFX Bullet Casing de donde hemos sacado nuestro objeto para proporcionar un punto de apoyo al raycast y a todos los scripts de las animaciones de disparo.

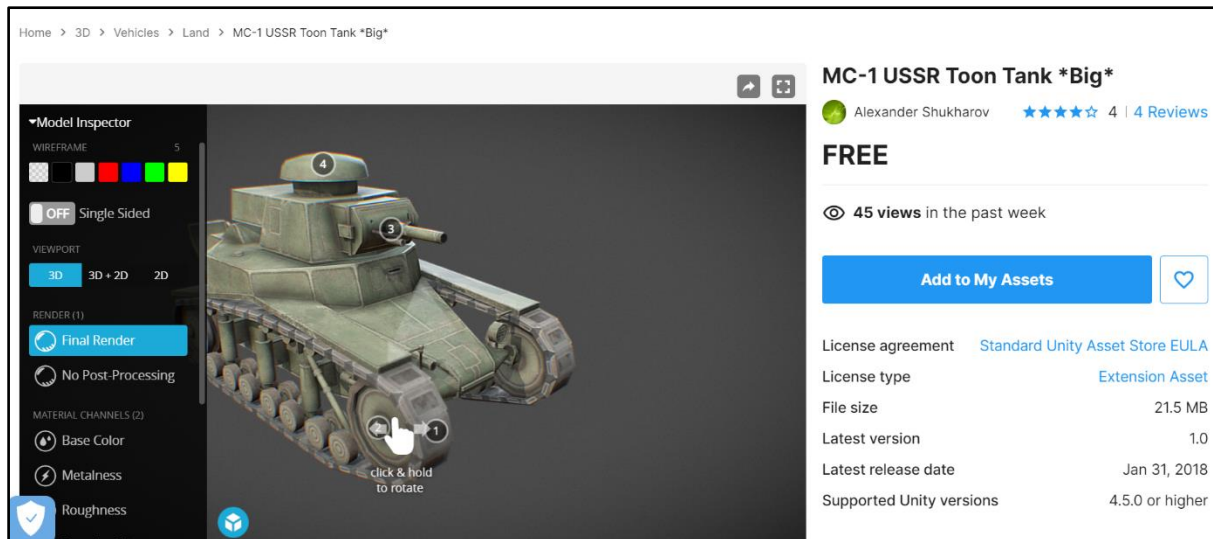
- *Unity Particle Pack 5.x:*



El asset encargado de las animaciones de las explosiones que hace cuando disparamos el tanque e impacta contra algún objetivo o el suelo.

Además de las explosiones utilizadas hay muchos otros efectos como granadas de humo, efectos de hechizos mágicos, de agua etc.

- *MC-1 USSR Toon Tank:*



Nuestro modelo de tanque utilizado al cual usamos como objetivo dentro del juego.

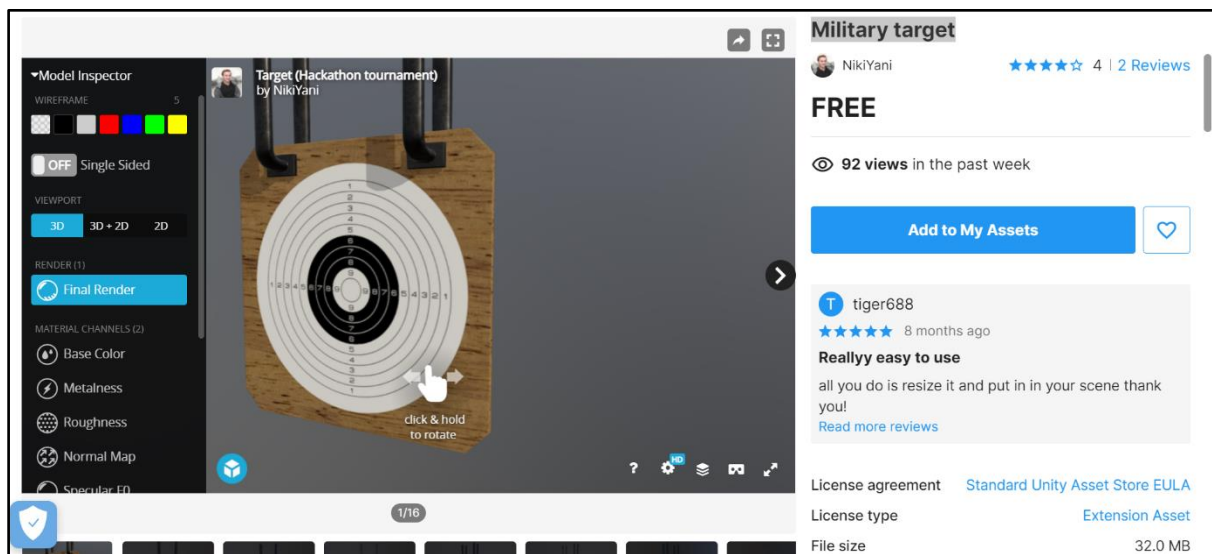
Este asset está basado en el primer tanque ligero creado por la URSS, se produjeron 960 unidades desde el año 1928 hasta 1931, participó en el conflicto *sino-sovietico* y en la segunda guerra mundial.



Sus dimensiones eran reducidas para ser un tanque 4,38 metros de longitud 1,76 de anchura y 2,10 de altura, gracias a estas reducidas dimensiones hacen que el peso fuese también bastante bajo en torno a las 5,9 toneladas.

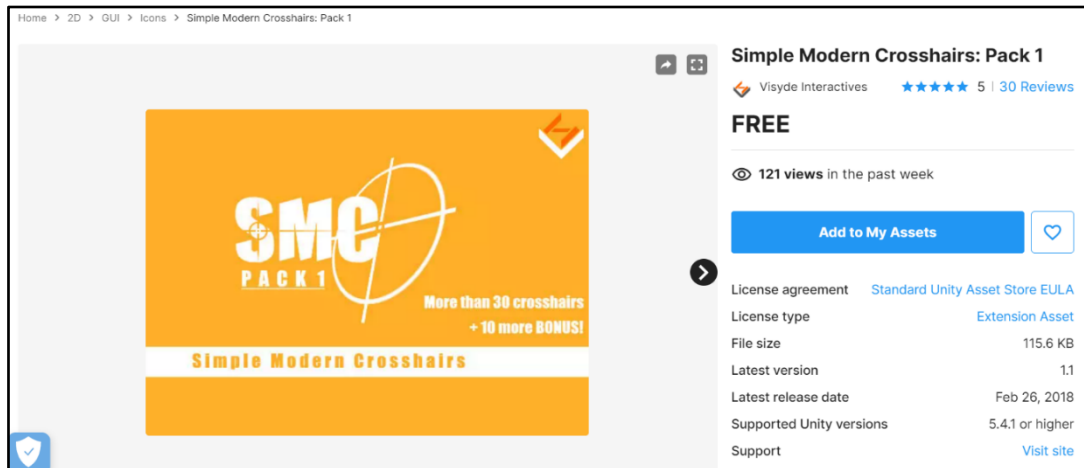
Contaba con un blindaje de 6-16mm y como arma principal un **cañón M1932(20-K) 45mm** y de secundaria dos **fusiles automáticos Fiódorov**.

- *Military target:*



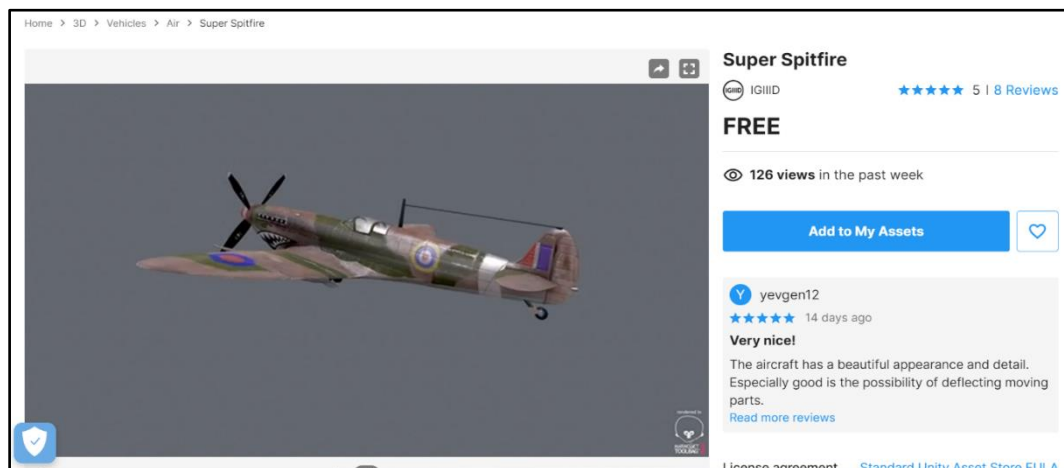
Nuestro modelo de diana militar usada en el juego como uno de los objetivos principales.

- *Simple Modern Crosshairs:*



Este Asset compondrá nuestra mira del tanque dentro del HUD que visualizamos desde la cámara del tanque en 1º persona.

- *Super Spitfire:*



Nuestro modelo de avión será usado como diana también dentro del juego, este será nuestro último objetivo junto con la diana y el tanque.

Está basado en un caza británico que estuvo activo durante la segunda guerra mundial, fue usado en países aliados.



Fue el único caza que pudo hacer frente a la temible aviación alemana. En cuanto al armamento dependía del tipo de alas que tuviera, las más comunes contaban con cuatro cañones de Hispano MK,V de 20mm o dos cañones de 20mm y cuatro ametralladoras ligeras.

- Uduino:

UDUINO
Arduino and Unity
The easy and flexible asset

Uduino Settings
Connection type: Desktop Serial
Desktop BLE
Android BLE
Android Serial
Web
General
Log Level: Error
Interface Type: None
Reading Method: Thread
Arduino
Baud Rate: 9600

Board Information
Last read message: uduinoidentity uduinoBoard
Last sent message: s132

Pin Mode Action Editor panel
13 Output HIGH LOW
A2 Input Read 0 128
5 Servo 63
6 PWM

Other settings
Discover pins Clear pins

Events
Advanced

Uduino - Arduino and Unity communication, simple, fast and stable
Marc Teyssier ★★★★★ 4 | 63 Reviews
€13.39
Seat - 1 +
Updated price and taxes/VAT calculated at checkout
224 views in the past week
Refund policy
Add to Cart
Secure checkout: VISA Mastercard PayPal Apple Pay Google Pay
the_cerberus ★★★★★ 4 months ago
Super Asset. Works as promised
I got some issues by setting up the Arduino and the

Uduino es el [asset más importante](#) que hemos utilizado, ya que es el que nos permite la comunicación con nuestra placa Arduino mediante una conexión wifi.

Esta biblioteca desarrollada por Marc Teyssier usa una red para conectarse a una placa wifi compatible con Arduino. Podemos acceder a todos los pines de dicha placa tanto en modo digital como analógico y en modo lectura o escritura.

Más adelante profundizaremos sobre esta librería y los usos que le hemos dado.

3.4 Vuforia

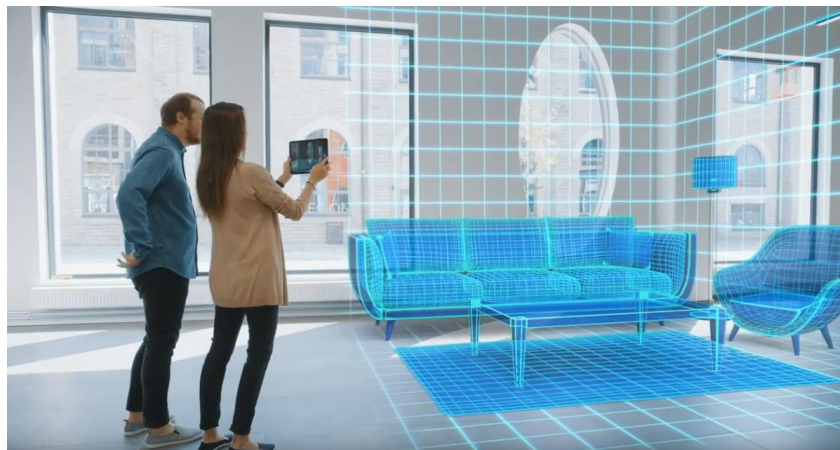
3.4.1 ¿Qué es Vuforia?

Vuforia es la plataforma de Realidad aumentada empresarial líder del mercado.



Se encarga principalmente de [mostrar los productos de la empresa en realidad aumentada](#) como una forma mejorada de marketing, por ejemplo, pueden hacer un escenario de realidad aumentada para ver como quedaría una oficina con unos muebles u otros.

Su herramienta [Vuforia Engine](#), en la versión que hemos usado nosotros, Vuforia Engine 9.8, es un motor de realidad aumentada capaz de ser incorporado como una librería a nuestro proyecto de Unity. Aunque ofrece soporte tanto en Android como en iOS.





3.4.2 ¿Como instalar Vuforia?


Buscamos en el navegador Vuforia Unity para que nos lleve directamente a [esta página](#), la cual nos da acceso para poder descargar Vuforia engine.


Vuforia Engine 9.8

Use Vuforia Engine to build Augmented Reality Android, iOS, and UWP applications for mobile devices and AR glasses. Apps can be built with Unity, Android Studio, Xcode, and Visual Studio. Vuforia Engine can also be accessed through the Unity Package Manager by adding Vuforia's package repository with the script below. Please make sure that Git is installed before running the script.

 [Add Vuforia Engine to a Unity Project or upgrade to the latest version](#)
add-vuforia-package-9-8-8.unpackage (3.04 KB)

 [Download for Android](#)
vuforia-sdk-android-9-8-5.zip (21.39 MB)

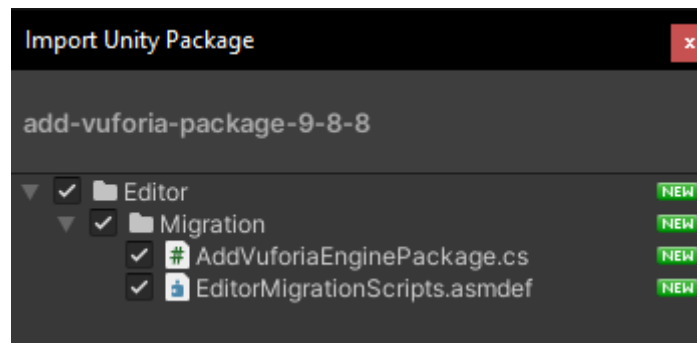
 [Download for iOS](#)
vuforia-sdk-ios-9-8-5.zip (67.47 MB)

 [Download for UWP](#)
vuforia-sdk-uwp-9-8-8.zip (20.37 MB)

[Release Notes](#)

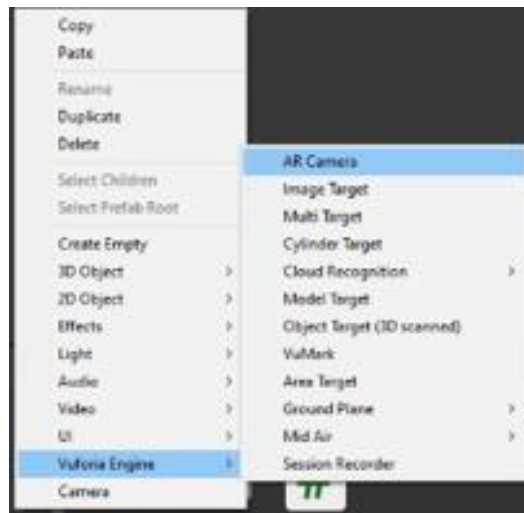
Vuforia te da [diferentes opciones de implementación](#) desde Android, iOS, descargar la aplicación de Windows, y la que nos interesa a nosotros, añadir Vuforia Engine a un proyecto de Unity.

Tras iniciar sesión en la plataforma empezará la descarga del instalador. Con un simple click en la librería esta se nos abre en nuestro proyecto (importante tener el proyecto abierto).



Se nos abrirá el [mismo menú de importar](#) que con cualquier cosa que nos descarguemos de la Asset Store, simplemente le damos a importar y lo tendríamos dentro del proyecto.

Una vez instalado completamente, para usar Vuforia en nuestro proyecto, debemos [eliminar la cámara](#) genérica que Unity genera al comenzar un nuevo juego.



A continuación, debemos incluir en el árbol de objetos una [AR Camera](#), disponible en el menú de la imagen.

En el menú de la derecha, el inspector de objetos podemos ver una gran cantidad de parámetros y scripts que podemos configurar.

Para comenzar, vamos a abrir la configuración de Vuforia, que se puede configurar en el botón del siguiente script.



Una vez dentro, vamos a incluir nuestro [numero de licencia](#), para ello vamos a ir a la web de Vuforia de nuevo. En el apartado “develop” nos ofrecerá la posibilidad de obtener una licencia gratuita, a la que le deberemos establecer un nombre y aceptar los términos y condiciones de uso.

Add a free Development License Key

You can change this later
License Name is required

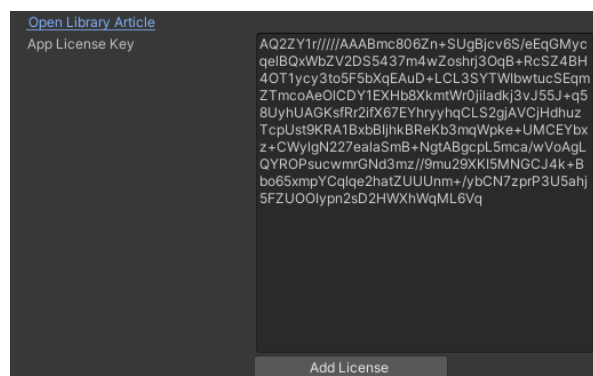
License Key

Develop
Price: No Charge
Reco Usage: 1,000 per month
Cloud Targets: 1,000
VuMark Templates: 1 Active
VuMarks: 100

☐ By checking this box, I acknowledge that this license key is subject to the terms and conditions of the [Vuforia Developer Agreement](#).

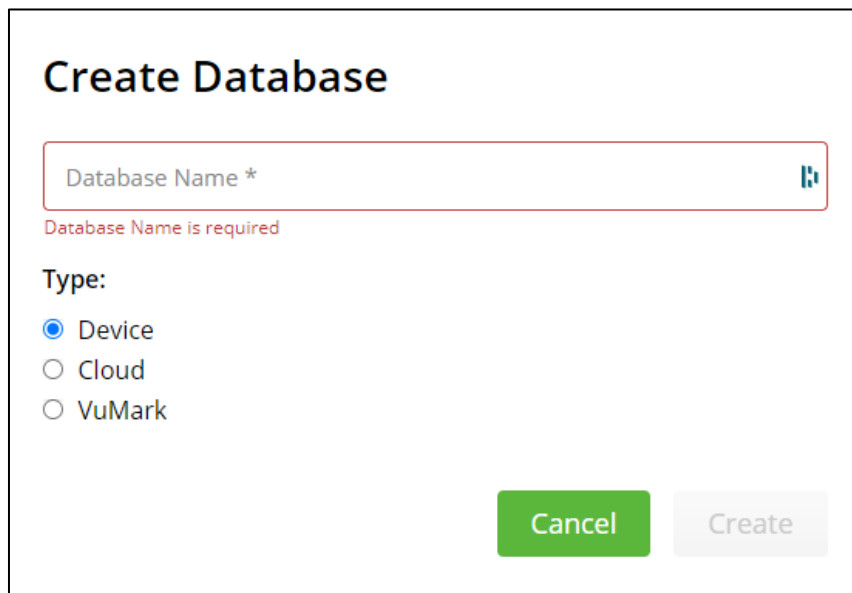
CancelConfirm

Una vez creada copiamos el largo número y lo añadimos en el siguiente apartado en Unity.



Vuforia nos ofrece la posibilidad de [integrar una base de datos](#) a nuestro proyecto, a estas bases de datos debemos añadir imágenes que corresponderán a un objeto dentro de Unity, que se instanciará cuando la cámara reconozca dichas imágenes, para explicarlo mejor, vamos a crear un ejemplo.

En la misma página donde hemos generado la licencia debemos ir al apartado “Target Images” donde al pulsar el botón de “Add Database”.



Create Database

Database Name *

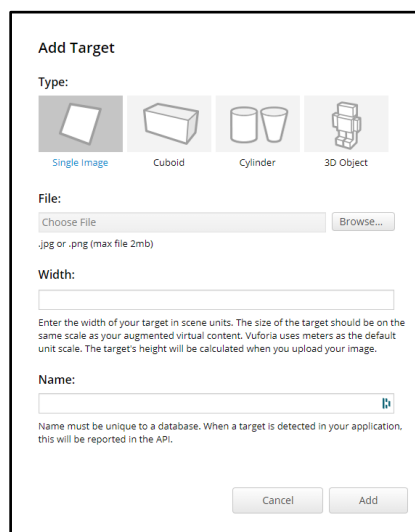
Database Name is required

Type:

- ☒ Device
- ☐ Cloud
- ☐ VuMark

Cancel Create

Una vez dentro de la base de datos debemos añadir algún target pulsando al botón de “Add Target”.



Add Target

Type:

- Single Image
- Cuboid
- Cylinder
- 3D Object

File:

Choose File Browse...

.jpg or .png (max file 2mb)

Width:

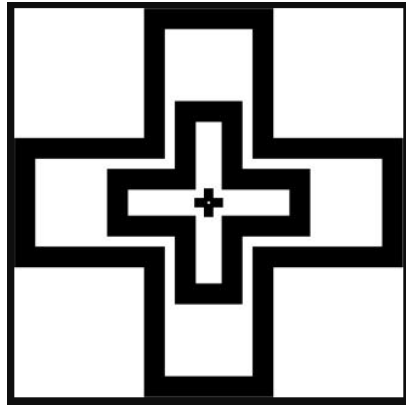
Enter the width of your target in scene units. The size of the target should be on the same scale as your augmented virtual content. Vuforia uses meters as the default unit scale. The target's height will be calculated when you upload your image.

Name:

Name must be unique to a database. When a target is detected in your application, this will be reported in the API.

Cancel Add

Completamos todos los campos, tipo, ancho de la imagen, nombre y un archivo de imagen, como por ejemplo esta:



Es recomendable que sea una imagen de este estilo para que Vuforia pueda reconocerla más fácilmente. Una vez creada, nos descargamos la base de datos, y la añadimos a Unity como si de un asset se tratara.

A continuación, crearemos un objeto “[Image Target](#)” dentro de Unity que debe tener como padre la AR Camera que hemos creado antes. También en el siguiente script debemos indicar que la imagen es de una base de datos, y seleccionar la que hemos creado antes.

A screenshot of the Vuforia Engine interface. At the top, there is a link that says "Download new Vuforia Engine version: 9.8.8". Below this, there are three dropdown menus. The first is labeled "Type" and has "From Database" selected. The second is labeled "Database" and has "Prueba" selected. The third is labeled "Image Target" and has "4mKKA" selected. Below these dropdowns is a button labeled "Add Target".

Ahora si creamos un objeto dentro de Unity, que tenga como padre la AR Camera, cuando se ejecute el juego, si la cámara detecta la imagen de la base de datos, aparecerá dicho objeto dentro del juego.

Como hemos visto el plugin de Vuforia ofrece una gran versatilidad y podemos utilizar este potente motor de realidad aumentada para proyectos muy grandes, sin embargo, nosotros nos hemos limitado a implementarlo para obtener la imagen de una cámara y utilizarla como escenario en Unity.



3.5 Arduino

Arduino es una [plataforma de creación electrónica de código abierto](#), que permite crear una infinidad de proyectos debido a que cuyo hardware y software son de software libre, además gracias a la comunidad hay una gran cantidad de contenido, documentación e ideas para aprender a utilizarlo.



Una de sus principales ventajas es el **precio**, ya que sus microcontroladores son bastante asequibles. Por otra parte, existen una gran cantidad de “**accesorios**” como pantallas LCD, controladores de motores o receptores Bluetooth, distribuidos por diferentes fabricantes, de forma que puedes realizar todo lo que te puedas imaginar

Arduino nace en Italia en 2005 a partir del **proyecto Wiring**, una plataforma creada por estudiantes para hacer frente al elevado precio que presentaban las placas **Basic Stamp** que usaban.



Hernando Barragán desarrolló la placa Wiring, su plataforma y su lenguaje de programación. Más tarde Massimo Banzi junto con David Mellis trabajaron para hacer la placa más ligera y económica, quienes finalmente iniciarían el proyecto de Arduino.

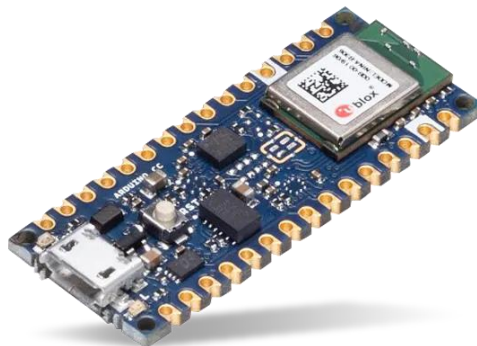
Actualmente existen varios tipos de placas que se ajustan a las necesidades de los diferentes usuarios, a continuación, vamos a ver las más usadas:

- La placa **Arduino Uno** es la más popular, tiene el tamaño de un carnet, dispone de un total de 19 pines de comunicación, 5 analógicos y 14 digitales. Requiere de 5v para funcionar por lo que se puede alimentar mediante USB.

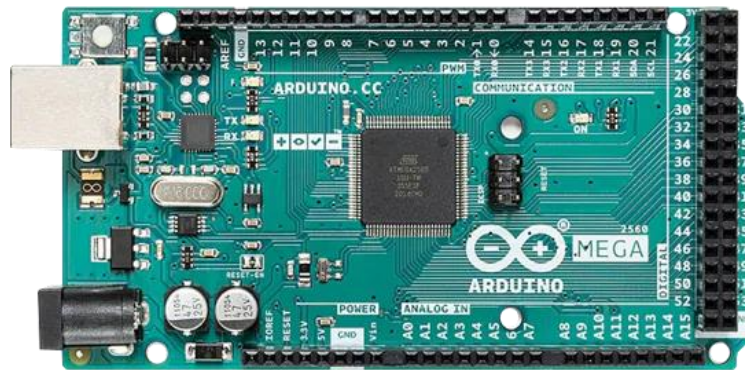
Aunque no es demasiado potente, es la placa perfecta para empezar, ya que es barata y siempre suele estar acompañada de kits de iniciación.



- [Arduino Nano](#) es una placa diseñada para utilizarse en proyectos donde cada centímetro cuenta, es una versión miniaturizada de la placa UNO por lo que posee el mismo procesador ATMEGA328P con 32Kb de memoria. Uno de los posibles usos de esta placa es en drones, debido a su reducido tamaño y peso.



- [Arduino Mega 2650](#) es la hermana mayor, integra el procesador ATmega2560, una memoria flash de 256 kB, por lo que podrá almacenar programas hasta 4 veces más grandes. 8 KB de RAM y 70 pines, entre analógicos y digitales. Uno de sus usos más frecuentes debido a sus características superiores es en impresión 3d.

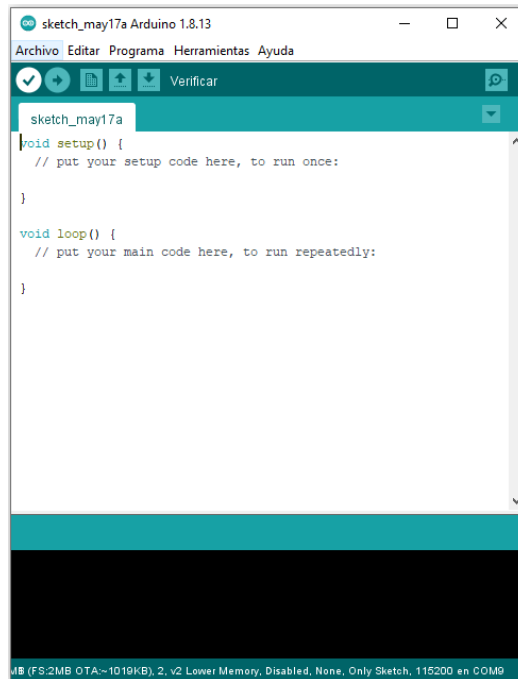


Esta es la placa que vamos a utilizar, ya que es una que teníamos de antes, podríamos utilizar también una placa Uno o incluso una nano, aunque la placa mega nos facilitará el trabajo debido a su gran cantidad de pines digitales y **PWM** disponibles.

3.5.1 Configuración del IDE Arduino:

Para comenzar a usar Arduino primero debemos configurar una serie de elementos:

Este es el [IDE de Arduino](#) donde vamos a programar. Para configurar nuestra placa debemos conectarla a nuestro pc para que se instalen los drivers necesarios automáticamente.



Una vez instalados podemos acceder a nuestro IDE y configurar nuestra placa, tan solo debemos ir al apartado “[Herramientas](#)” y en la sección Placa debemos introducir nuestro tipo de placa, en nuestro caso Mega 2650, en la sección puerto debemos seleccionar en que puerto está conectado nuestra placa.

Placa: "Arduino Mega or Mega 2560"	>
Procesador: "ATmega2560 (Mega 2560)"	>
Puerto: "COM7 (Arduino Mega or Mega 2560)"	>

Para comprobar si nuestra placa está lista para usar vamos a [cargar el programa](#) que viene escrito por defecto, aunque no hace nada, nos servirá para saber si nuestra placa funciona correctamente.

Si pulsamos el botón del “[tick](#)” verificaremos que el programa no tiene ningún error. El segundo botón compilará el programa y lo subirá a la placa, en la parte inferior nos aparecerá una barra de carga y cuando haya terminado, nos informará con un mensaje de que el programa ha sido subido.



En caso de que haya algún error aparecerá un mensaje con un cuadro naranja. La mejor solución es buscar el código de error en internet, ya que hay muchos foros donde nos pueden ayudar.

A los programas en Arduino se les denomina [sketch](#). Los sketch son archivos .ino que contienen el código de nuestros proyectos.

El lenguaje de programación de Arduino es una adaptación de [C++](#) derivada de [avr-libc](#), un proyecto que provee de una serie de bibliotecas de [C](#) para poder usarlo con microcontroladores.



Respecto al código de Arduino, en cada programa tenemos dos métodos muy importantes:

- [Setup\(\)](#): este método es llamado en el momento en el que se ejecuta nuestro programa. Solo se va a ejecutar una sola vez, se usa principalmente para inicializar variables, declaración de bibliotecas o los modos de un pin (output o input).

- `Loop()`: este es el bucle principal que se va a ejecutar durante todo el tiempo que el programa se esté ejecutando.

En este [enlace](#) podemos ver los métodos más importantes que incluye la biblioteca de Arduino.

3.5.2 Componentes Arduino

Arduino es compatible con miles de productos electrónicos con los que se pueden realizar una cantidad inimaginable de proyectos diferentes, desde motores hasta leds pasando por pantallas LCD. A continuación, vamos a explicar los que hemos utilizado en nuestro proyecto.

Alimentación:

Para el proyecto hemos utilizado dos baterías LiPo (polímero de iones de Litio), este tipo de baterías funcionan igual que las baterías de iones de litio, extensamente utilizadas como en las baterías de los móviles. El funcionamiento de una batería LiPo es el siguiente: (Wikipedia)

“el intercambio de electrones entre el material del electrodo negativo y el material del electrodo positivo mediante un medio conductor. Para evitar que los electrodos se toquen directamente, se coloca entre ellos un material con poros microscópicos que permite tan solo los iones (y no las partículas de los electrodos) migren de un electrodo a otro.”

Este tipo de **baterías son ligeras y pueden almacenar una gran cantidad de energía**. Están compuestas por celdas, cada una de las cuales tiene un voltaje nominal de 3.7V y 4.2V.

Nuestras baterías son 3s, lo que significa que tienen 3 celdas, lo que le permite tener un voltaje nominal de entre 11.1V y 12,6V. También tienen una capacidad de 3500mAh, haciendo un total de 7000mAh.

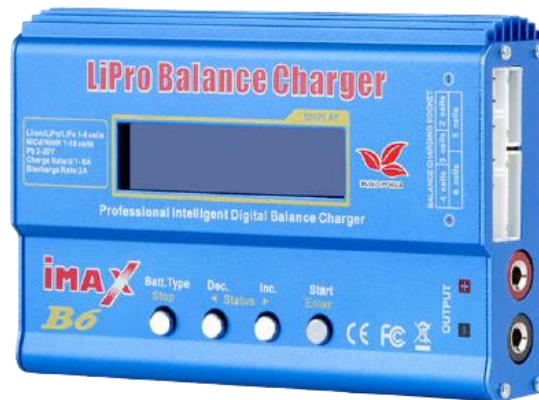
El proyecto en total tiene un consumo estimado de 2870mA:

- **Placa Arduino Mega 2650** -> 93mA.
- Módulo emisor de radio **DVR VTX + Cámara** -> 250mA
- **Motores 795** -> 1A cada uno a mitad de potencia.
- Servomotor **sg90** -> 100mA.
- Stepper motor **28BYJ-48 + ULN2003** -> 150Ma.
- Modulo Wifi **NodeMCU 12E** -> 170Ma.

Con este consumo, las baterías, durarían unas **2,5h**. (Los consumos son estimados puesto que los motores no estarían funcionando a carga máxima ni constantemente).

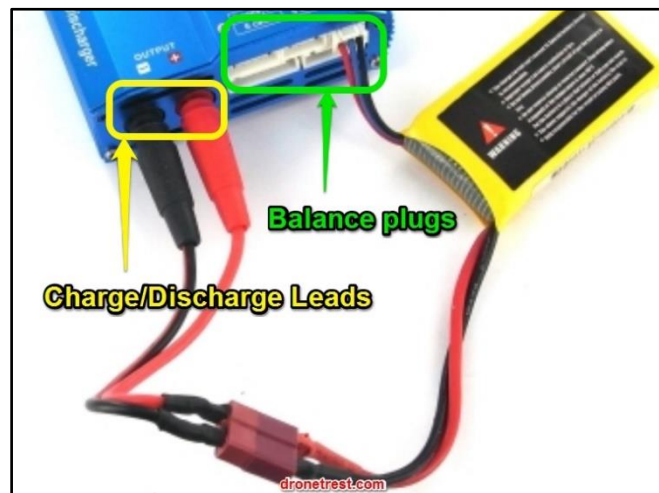
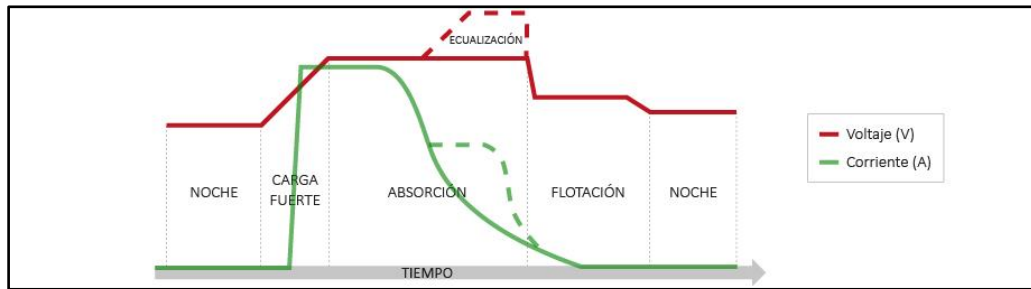


Este tipo de baterías requieren ciertos cuidados al ser **bastante delicadas**, sobre todo en la fase de carga, ya que pueden incluso arder. Por eso nosotros utilizamos un cargador balanceador.



Los **balanceadores** de carga realizan una carga equilibrada entre todas las celdas de la batería. Durante este proceso verifica todos los voltajes individuales cada celda y se asegura de que tengan todas el mismo, ya que si una de ellas recibe un voltaje por encima de su capacidad podría dañarse e incluso incendiarse. Además, este aparato ofrece una carga completa de todas las etapas necesarias para alargar la vida de la batería.

Esta gráfica representa las **fases para realizar una carga** de la forma más segura posible. Este es el esquema de conexión para la carga:



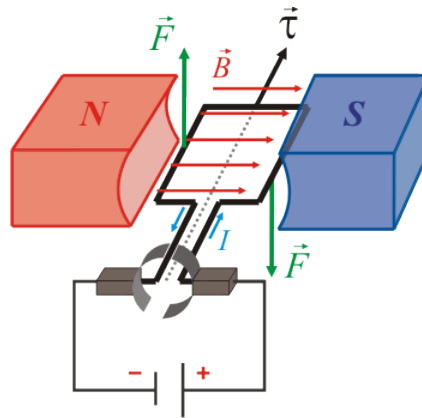
Motores:

Para simular el movimiento de un tanque en nuestra maqueta hemos utilizado **diferentes tipos de motores** que se adaptan mejor a las necesidades de cada situación, por sus diferencias entre torque, potencia o precisión.

Un motor eléctrico de corriente continua es un componente electromecánico que permite transformar la energía eléctrica en energía cinética en forma de rotación.

Su funcionamiento se debe a la interacción entre un campo magnético y un campo eléctrico.

Un **conductor eléctrico** experimentará una serie de fuerzas si las corrientes que circulan por el están en ángulo recto con un campo magnético, si se invierte la dirección de la corriente o del campo magnético se producirán fuerzas en sentido contrario.



Si una **corriente eléctrica** se desplaza por un cable, esta crea un campo magnético a su alrededor, entonces al colocar imanes alrededor del cable, este es atraído repelido por el campo magnético del imán, por lo que el cable comenzará a girar.

Este es el principio básico de funcionamiento de un motor de corriente continua.

Motores 795:

Para el movimiento de las orugas del tanque hemos utilizado un motor de tipo 795, un motor de corriente continua capaz de trabajar con una tensión de entre 12 y 24V, la corriente de entrada media sería de 1A aunque durante el arranque puede llegar a consumir hasta 10A, tiene un consumo máximo de 180W.



A [máxima potencia](#) puede llegar hasta 10.000 rpm con una fuerza de torsión de 4kg.cm. Aquí tenemos una tabla con todas sus características junto a otros tipos de motores para comparar.

Model	775	795	895 (low speed)	895 (high speed)
Rated voltage	12V	12V	12-24V	12-24V
Speed	12V 10000rpm	12V 11000rpm	12V 3000rpm 24V 6000rpm	12V 6000rpm 24V 12000rpm
Torque	3kg.cm	4kg.cm	12V: 5.2kg.cm 24V: 10kg.cm	12V: 4.2kg.cm 24V: 8kg.cm
Former high level	4.5mm	4.5mm	4.5mm	4.5mm
Front steps diameter	17.5mm	17.5mm	18mm	18mm
Body diameter	42mm	42mm	48mm	48mm
Shaft diameter	5mm	5mm	5mm	5mm
Body length	67mm	70.5mm	72mm	72mm
Length of output shaft at tail	6mm	3mm	6mm	6mm
Diagonal installation pitch	29mm	29mm	29mm	29mm
Mounting hole size	M4	M4	M4 M5	M4 M5
Power	150W	180W	200W	200W

La principal razón para su elección fue el precio, ya que no difería del precio del tipo 775 siendo algo más potente y considerablemente más económico que los tipos superiores, además la tensión máxima coincide con la tensión de nuestras baterías.

Para el proyecto hemos utilizado [dos de estos motores](#), uno para cada oruga, de esta forma para hacer un giro a la derecha basta con enviar potencia al motor de la izquierda como ocurre en los tanques reales. Para controlar estos motores debemos utilizar un driver compatible, en nuestro caso hemos utilizado el BTS 7960:

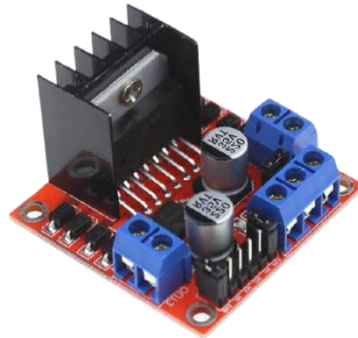
El BTS 7960 es un driver para motores de corriente continua que nos permitirá controlar uno de nuestros [motores 795](#) fácilmente. Este driver puede proporcionar hasta 43A de corriente con una tensión de entre 6V y 27V.

Además, incluye diferentes protecciones:

- [Temperatura](#): tiene un sensor de temperatura que, al llegar a un límite, provocará el apagado de los pines de entrada.
- La [corriente](#) se mide a través de dos interruptores, cuando esta llega al límite máximo uno de los interruptores se desactiva.
- [Sub tensión](#): Si el suministro de voltaje cae por debajo de los 5,4V el controlador se apagará para evitar un movimiento incontrolado del motor.

También cuenta con una parte lógica que funciona a 5V con la que podremos controlar el motor desde la placa de [Arduino](#) mediante [PWM](#).

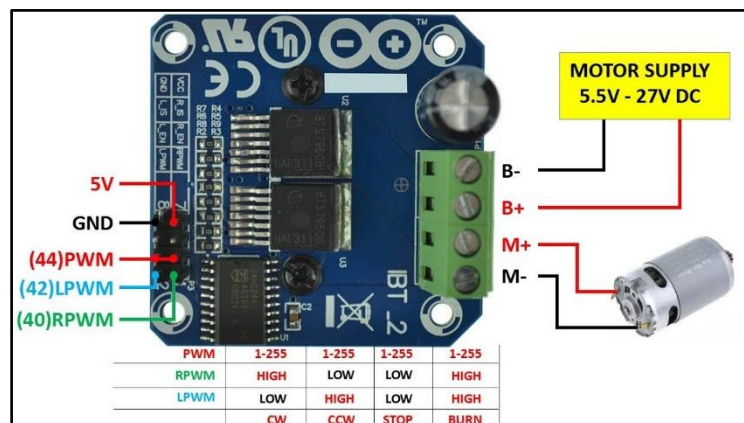
Existen drivers menos potentes y económicos como el driver **L298N**, pero debido a la limitación de 2A de este, tuvimos que elegir uno más potente, ya que nuestros motores para el arranque inicial pueden consumir un pico de 10A.



Este consumo, el cual desconocíamos, hizo que uno de estos módulos se quemara, ya que no tiene ningún tipo de protección.

El driver BTS 7960 tiene un coste de alrededor de 3€ siendo más caro que el driver L298N que se puede adquirir desde 1.5€.

El montaje del driver es **sencillo**, los pines RPWM y LPWM deben conectarse a las salidas de Arduino capaces de emitir señales PWM el siguiente esquema indica la dirección a la que girará el motor dependiendo de las señales que enviemos.



Los pines 5V y GND se conectarán a sus respectivas salidas de la placa de Arduino para alimentar la [lógica del driver](#).

El motor se conectará a las salidas M+ Y M- por las que lo alimentará con el voltaje y amperaje según se lo indique la lógica del driver.

Los pines B+ y B- servirán para [alimentar el motor](#) conectándole una fuente de alimentación capaz de entregar un voltaje de entre 5.5V y 27V de corriente continua.

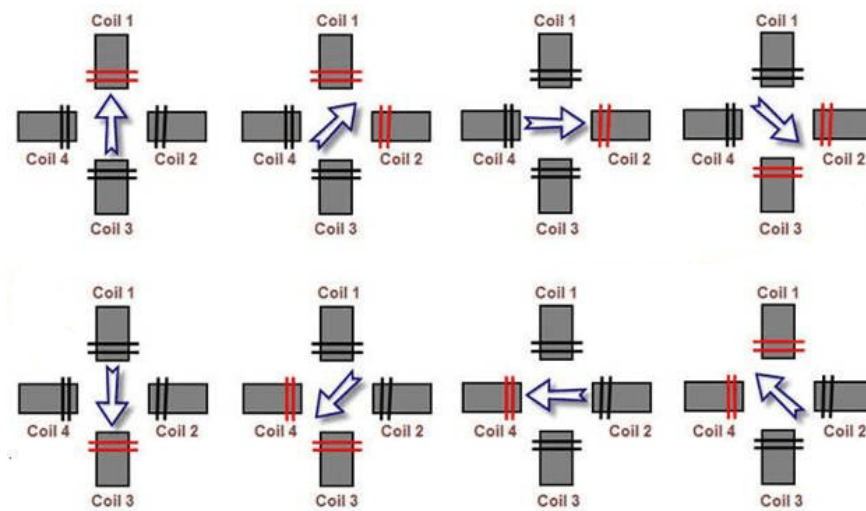
Motor de paso a paso:

Un motor de paso a paso funciona de manera distinta a un motor de corriente continua común. Estos están compuestos por un [rotor](#) que gira junto al eje gracias al campo magnético que genera un [estator](#).

El estator es un conjunto de bobinas dispuestas alrededor del rotor, si la corriente fluye a través de este se crea el campo magnético que hace que gire el rotor.

De esta forma dependiendo de las bobinas que se energicen del estator el rotor se moverá a una posición u otra.

Este es un esquema del funcionamiento de un motor de paso a paso:



Este tipo de motor se caracteriza por tener un **gran torque** y una muy buena **precisión**, gracias a estas características hemos decidido que su mejor desempeño va a ser girar la torreta superior. Este trabajo lo va a realizar un pequeño motor 28BYJ-48 como el de la imagen:



Además del motor, vamos a necesitar el driver ULN2003:

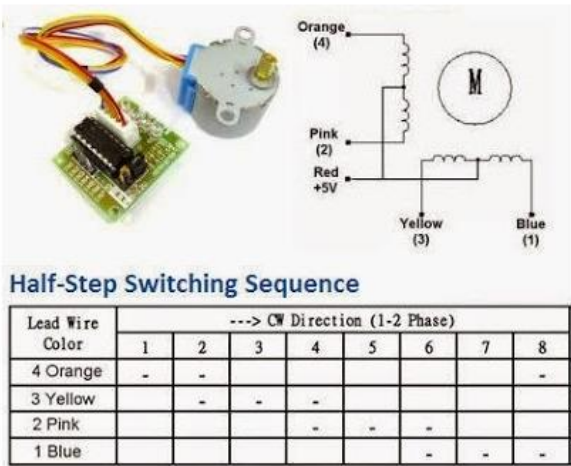
Es un driver especialmente diseñado para controlar **motores paso a paso**, aparte de eso puede ser utilizado para manejar relés o cualquier carga de corriente continua de bajo consumo.

Su dimensión es de 35 x 33 x 11mm, peso de 42g, es compatible con los motores de paso a paso de 5V Y 12V de 5 cables, integra un jumper que permite el paso del voltaje hacia el motor.

La conexión entre el módulo y el motor es bastante intuitiva ya que cuenta con un conector con ranuras para evitar posibles errores a la hora de conectarlo.

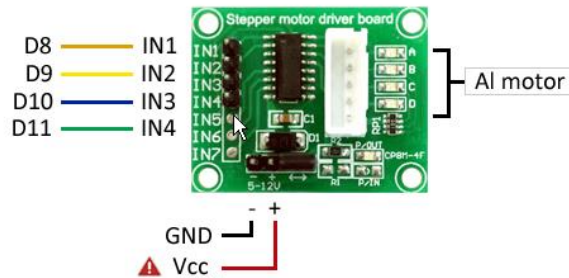


En este módulo existen cuatro leds que indican que bobina se está usando.



Para la alimentación de este módulo se recomienda el uso de [baterías externas](#) como en nuestro caso y no usar la salida que tiene la placa de Arduino ya que es necesario más corriente de la que nos puede ofrecer.

Los pines desde el IN1 hasta el IN4 son los que serán conectados a las salidas digitales del Arduino.



Servomotor:

Un servomotor es un tipo de motor de corriente continua que otorga una mayor precisión con un consumo de energía mucho más pequeño, además, tiene mejor torque que un motor convencional.

La principal característica de un servomotor, aparte del alto par, es [el control de posición](#), que nos permite elegir la posición que queremos que tome y la posición que tiene en cada momento.

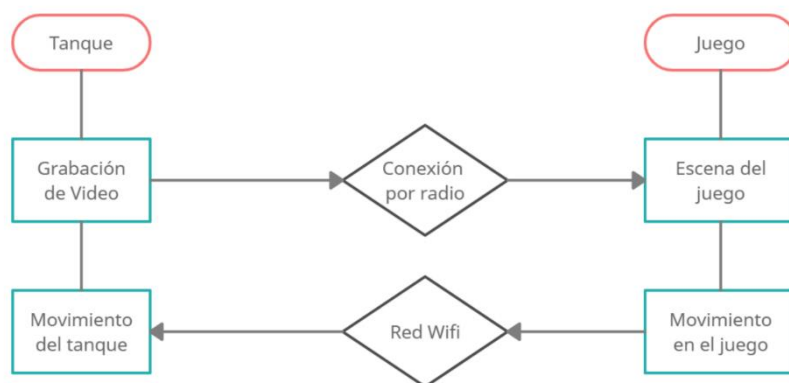
Un servomotor consta de varias partes, un motor eléctrico convencional, que proporciona el movimiento del eje, un sistema de control, que permite controlar el motor mediante señales **PWM**, un conjunto de engranajes, unidos al eje, permitiendo aumentar el par, y un potenciómetro, que permite saber en todo momento la posición en la que se encuentra el eje del motor.

Nuestro servomotor será un **SG90**, este servo nos permite tener un ángulo de giro de 180°, que van desde -90° hasta 90°, como su función va a ser subir el cañón o bajarlo, no nos va a hacer falta un motor que haga giros completos.



Conectividad:

Un aspecto fundamental para nuestro proyecto es la comunicación de los componentes de Arduino con Unity. Hemos utilizado **dos vías** diferentes, wifi y radio, este sería un ejemplo del esquema de conexión:



Conexión wifi:

Para la conexión entre unity y Arduino comenzamos usando una conexión [bluetooth](#) con el módulo [hc-06](#), pero debido a sus limitaciones(rango de señal y velocidad de la transmisión de información), decidimos hacer un gran cambio e implementar una conexión wifi aprovechando que en Unity hay un asset llamado Uduino que permite una integración sencilla entre ambos.

El módulo que utilizamos fue el [NodeMcu 1.0](#), que utiliza el chip ESP8266 desarrollado por la empresa Espressive específicamente para el desarrollo del IoT (internet de las cosas).

Tiene una CPU RISC de 32 bits a 80Mhz, tiene pines UART para la transmisión de datos de datos y usa el estándar IEEE 802.11.



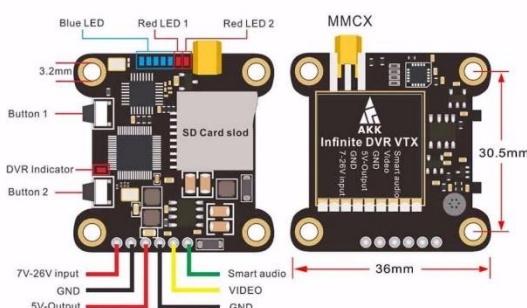
Este chip por si solo tiene la [potencia suficiente](#) como para sustituir la placa de Arduino que hemos utilizado, pero debido a la falta de pines de la NodeMcu decidimos hacer que esta placa sirviera como intermediario entre Unity y la placa de Arduino Mega 2650, de forma que la NodeMcu recibe la información que manda Uduino de Unity por wifi, y esta envía [a través de 4 cables](#) señales digitales, de forma que podemos enviar 4 bits al mismo tiempo.

Conexión por Radio:

Para la **emisión de video** hemos utilizado una tecnología similar a la que es usada en los drones **FPV** (first person view), que se utiliza para controlar drones a vista de pájaro de forma que son posibles vuelos de mucho más alcance y precisión.

Este tipo de drones utiliza un emisor de radio que emite en un rango de entre **5650 y 5925MHz**. La principal ventaja del uso de ondas de radio es la baja latencia, puesto que estas pueden viajar a la velocidad de la luz (ambas son un tipo de radiación electromagnética).

Nuestra emisora de radio es una DVR-VTX de la marca AKK:



Esta emisora tiene un lector de tarjetas SD, de forma que podemos grabar videos y guardarlos a muy buena calidad. Cuenta con un conector para antenas donde nosotros hemos colocado una antena omnidireccional, de forma que la señal se transmite en todas las direcciones.

La corriente de entrada será de entre 7 y 27V y cuenta con una salida de 5V que podemos aprovechar para alimentar nuestra cámara.

Para la **recepción de la señal de radio** hemos utilizado un ROTG-01, este gadget permite recibir la señal de radio y transformarla en video, que pasa al ordenador como si de una webcam se tratara.

Cuenta con un conector para instalar una [pequeña antena](#) y ampliar el rango de búsqueda de señales, cuenta con un botón con el que realizaremos un barrido de todos los canales y cubrir todas las bandas de frecuencia a 5.8GHz.



Cuenta con una [latencia de unos 100ms](#), aunque no es lo ideal para [FPV](#), puesto que una latencia tan alta puede llegar a marear, es perfecto para nuestro proyecto, ya que es económico y fácil de usar.

Necesita una alimentación de 5V por lo que basta con conectarlo por USB.

Tanto emisor como transmisor [cuentan con un chip que codifica/descodifica](#) la imagen con métodos diferentes. La codificación de video comprime la imagen para que ocupe el menor espacio posible eliminando partes redundantes del video. Los métodos para realizar estas compresiones se denominan [codecs](#) de video como por ejemplo MPEG-1 y 2 O H264.

El receptor de video incluye el estándar de video UVC (USB Video Class) que permite transmitir video por USB.

4. Objetivos a cumplir

- Creación de un tanque funcional físico.
- Implementación de los diferentes servomotores y la placa de Arduino para su correcto funcionamiento.
- Envío de imagen en vivo al ordenador, a parte, de controlar por wifi el tanque en tiempo real.
- Implementación de Vuforia dentro de nuestro juego .
- Creación de enemigos dentro de nuestro campo de batalla (Dianas, Aviones, Tanques).
- Sistema de puntos para contabilizar como de bien se ha dado la partida.
- Implementación de un HUD en el juego para simular una mira del tanque junto con una animación de recarga.
- Animación y disparo del tanque mediante RayCast.
- Sistemas de vida para los diferentes enemigos.
- Movimiento de los enemigos por el campo.
- Movimiento del tanque simultáneamente dentro del juego y en la realidad.

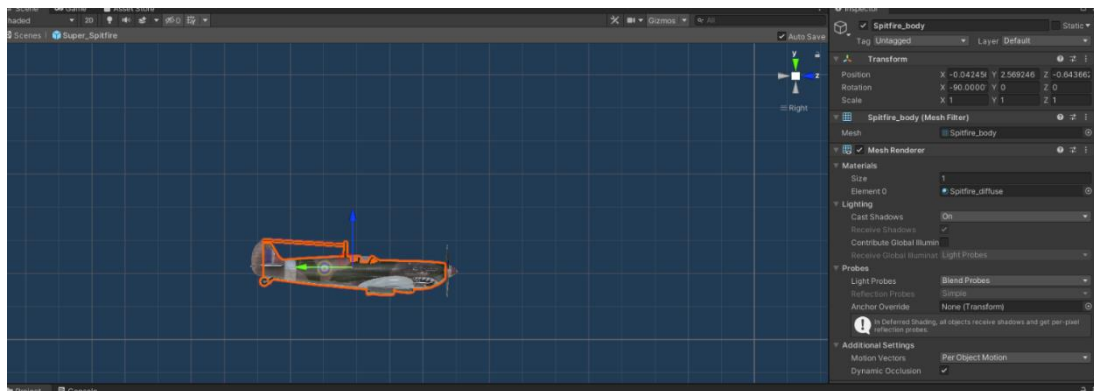
5. Desarrollo

En este apartado vamos a ver las principales clases ejemplos de código que hemos utilizado, además de su funcionamiento y su explicación.

5.1 Unity

GameObjects:

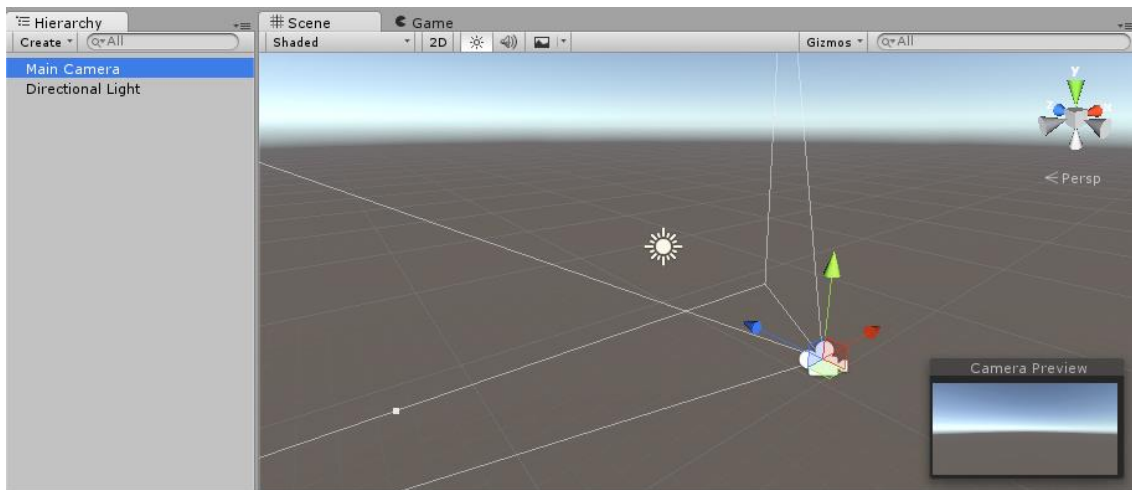
Los GameObjects son los **objetos más utilizados e importantes** de Unity. Todo en Unity son objetos desde la cámara, el personaje etc. Se diferencian uno de otro gracias a sus propiedades, es la clase base usada por **todas las entidades** de Unity.



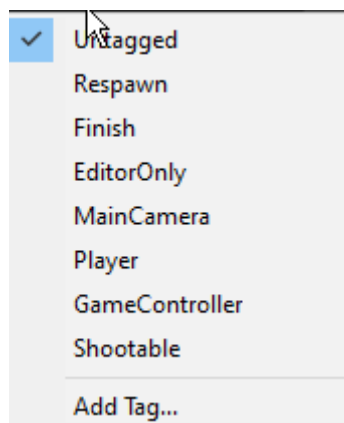
En los **GameObjects** podemos encontrar distintos atributos (características propias) como podrían ser:

- **activeInHierarchy**: Se encarga de definir el estado de nuestro GameObject en la escena, activo/inactivo.
- **activeSelf**: Nos dirá si el objeto esta activo o inactivo.

- **layer:** Se encarga de asignar una capa a nuestro GameObject, esto nos puede ser útil a la hora de ocultar objetos que se encuentren en una capa determinada.
- **scene:** Asigna a nuestro objeto a una escena de nuestro proyecto, cada escena contiene todos nuestros Game Objects y es donde podremos agregarlos a nuestro videojuego.



- **tag:** Etiqueta que se puede asignar a nuestros GameObjects para que podamos identificarlos de una manera más sencilla, Unity tiene algunos tags predefinidos además de poder crear cualquier etiqueta personalizada.



- **transform:** se encarga de regular la rotación (hacia donde mira) la posición (lugar de la escena donde se

encuentra) y la escala(tamaño). Esta propiedad es muy útil para cambiar la escala la posición o la rotación de nuestro objeto por código, asimismo, se puede saber estos valores en cualquier momento a través de código con los métodos para saber la escala de un objeto Unity con los métodos `transform.position()`, `transform.rotation()` y `transform.localScale()`.

Dentro de los `GameObject` podemos encontrar muchos métodos que nos pueden ser de gran utilidad al crear nuestros juegos vamos a enumerar los más importantes:

- **AddComponent**: Nos permite añadir componentes a nuestro `GameObject` a partir de nuestro código (`transforms`, `Rigidbody`, `Colliders...`), esto nos puede ser de gran utilidad cuando queramos añadir componentes en un momento exacto de la ejecución del código.
- **GetComponent<Type>**: Nos devuelve el objeto del tipo especificado lo cual nos ayuda a ejecutar métodos que se encuentren en un script asociado a un objeto, obtener componentes como `colliders` etc.
- **SetActive**: Permite cambiar el estado de activo a inactivo mediante un booleano.
- **TryGetComponent**: Nos devuelve un booleano dependiendo si tipo de objeto que hemos especificado existe en el `GameObject`, lo que nos permite evitar errores a la hora de ejecutar nuestro código ya que si intentamos obtener un tipo de componente que no se encuentre en el objeto nos enviará un mensaje de error.

Métodos de control de objetos:

Instantiate:

Método que se encarga de crear un clon de un gameObject desde un script. Estos clones tendrán las mismas componentes: valores de posición, rotación y escala, hitbox... del GameObject a partir del cual han sido creados. Estos objetos funcionan de manera independiente.

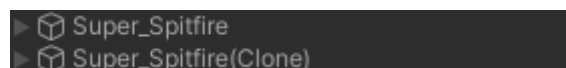
Su declaración es:

Instantiate (**Object** objetoOriginal, **Vector3** posición, **Quaternion** rotación, **Transform** objetoPadre);

Los parámetros que se pueden incluir son:

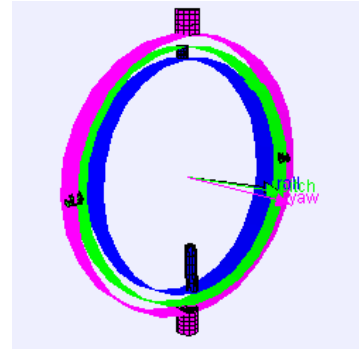
- **objetoOriginal:** GameObject existente el cual queremos copiar, es el único parámetro obligatorio.

El objeto creado tendrá el nombre del objeto inicial más un "(clone)" al final, de manera que es mucho más fácil identificarlo.



- **posición:** Vector de tres posiciones (de dos si es un videojuego en 2D) que determinará el lugar de aparición de nuestro clon, en el caso de que sea null el objeto aparecerá en las coordenadas (0,0,0).

- **rotación:** Vector de tres posiciones que se encarga de representar las rotaciones, este tipo de vectores se les denomina Quaternion, están formados por números complejos, gracias a ellos se elimina el problema del **bloqueo cardán** (pérdida de un grado de libertad en un objeto con tres rotores, que ocurre cuando los ejes de dos o más rotores se colocan de manera paralela bloqueando la rotación).



- **objetoPadre:** GameObject al cual queremos que sea asignado nuestro clon.

Destroy:

Método encargado de la destrucción de GameObjects.

Su declaración es:

Destroy(**GameObject** objeto, **float** time);

Los parámetros que se pueden incluir son:

- **objeto:** GameObject que deseamos destruir.
- **time:** Numero decimal el cual va a indicar el tiempo tras el cual el objeto va a ser eliminado del videojuego, en el caso de que sea 0 o null el objeto se eliminara inmediatamente.

RayCast:

Para hacer el disparo del tanque usaremos la clase `raycast`, que implementa Unity por defecto. Consiste en el lanzamiento de un `rayo invisible` desde un punto de origen, el raycast nos informará si choca con algún objeto, lo que nos permitirá eliminar los objetos con los que choquemos, reducir su vida etc.

Nos devolverá un booleano el cual será verdadero `si ha chocado` con algo y falso si no.

Puede usarse tanto en Unity 3D como en 2D, suele usarse en juegos de disparos, también puede usarse en la inteligencia artificial de algunos `NPC (Personaje no jugador)`, ya que nos permitirá que los personajes controlados por la maquina no se vayan chocando con las paredes, ya que con este rayo puede detectar si tiene alguna pared cerca y cambiar su dirección.

Para la creación de este objeto necesitaremos varios parámetros:

`Raycast(Vector3 origen, Vector3 direccion, out RaycastHit informaciondelgolpe, float rango, int excepciones, QueryTriggerInteraction desencadenantes);`

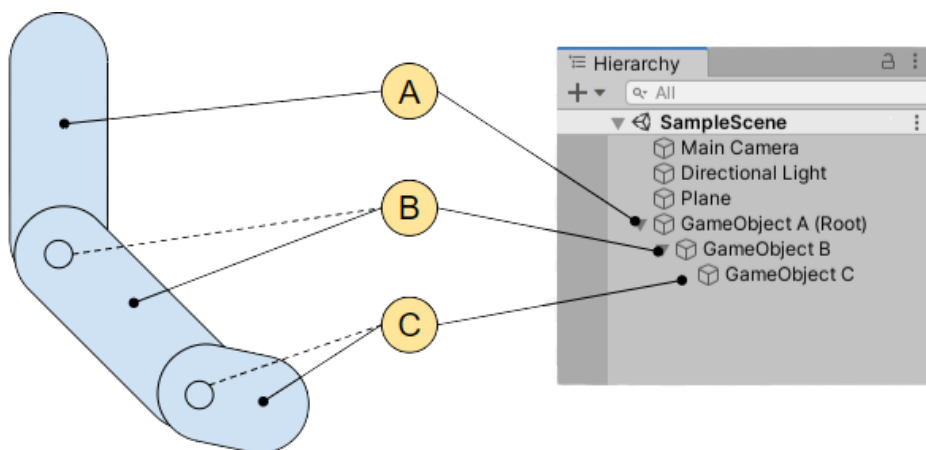
- **Origen:** Vector de tres coordenadas donde se definirá el origen del rayo
- **Dirección:** Vector de tres coordenadas donde se definirá la dirección en la que ira el rayo.
- **Informaciondelgolpe:** En el caso de colisionar con algún objeto nos dará más información de la colisión.

- **Rango:** Valor numérico que definirá el rango del rayo.
- **Excepciones:** Se utiliza para ignorar selectivamente algunas colisiones del rayo.
- **Desencadenantes:** Código que se ejecutara en el caso de que se golpee algún objeto.

RaycastHit:

Objeto utilizado para almacenar la información del golpe del Raycast entre los métodos del RaycastHit podemos encontrar:

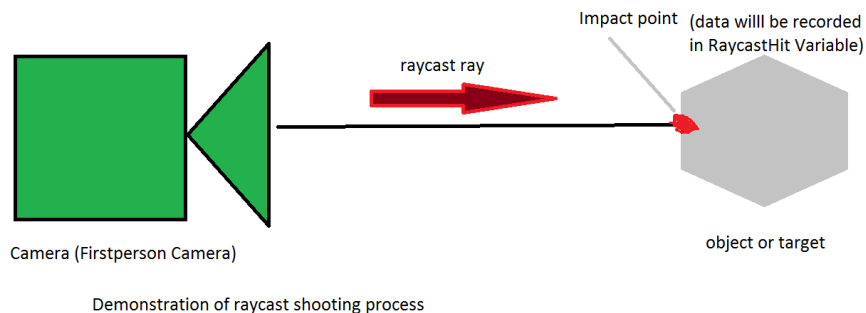
- **ArticulationBody:** Devolverá un ArticulationBody (permite la creación de articulaciones físicas con GameObjects que están estructurados jerárquicamente, que ayudan obtención de movimientos mucho más complejos) del objeto golpeado.



- **Collider:** Devolverá el objeto collider (elemento que envuelve nuestro objeto que permite la detección de golpes, los collider pueden tener cualquier tipo de forma, aunque se recomienda que sea lo más ajustado a nuestro GameObject) del objeto golpeado.

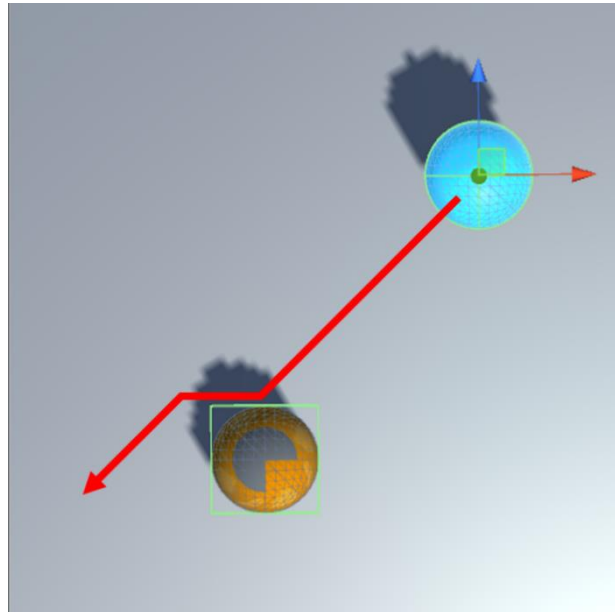


- **Distance:** Dará un numero decimal el cual nos indicará la distancia desde el punto de inicio del rayo y el punto de colisión.
- **lightmapCoord:** Devolverá la coordenada del mapa de luz ultravioleta.
- **Point:** Devolverá el punto donde el rayo impactó con el objeto.



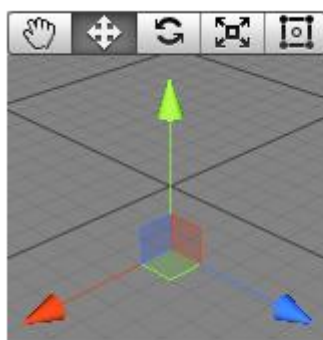
- **RigidBody:** Devolverá el rigidbody(permite que un gameobject tenga físicas, como la gravedad, y permite que reaccione a impactos) del objeto el cual se ha impactado.

- **Transform:** Devolverá el objeto transform del objeto impactado y de esta manera se podrá modificar sus valores.

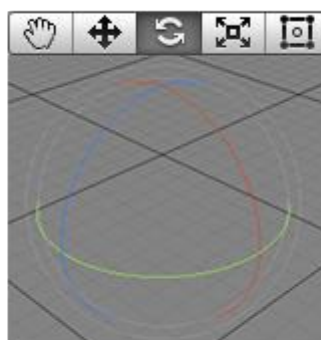


Transform

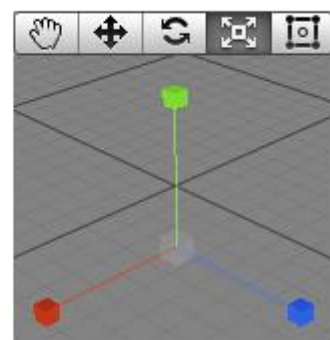
Se encarga de la modificación de la posición, rotación y escala de un GameObject, todos los objetos de un videojuego incluyen uno.



Translate (W)

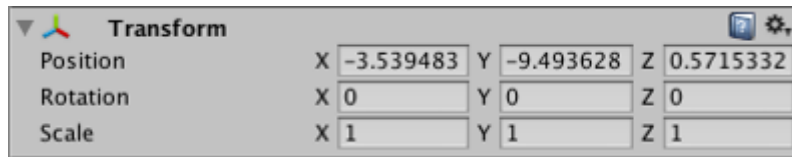


Rotate (E)



Scale (R)

Estos valores son tomados relativamente desde el objeto padre del Transform, en el caso de que no tenga padre estos valores se tomaran en referencia a la escena.

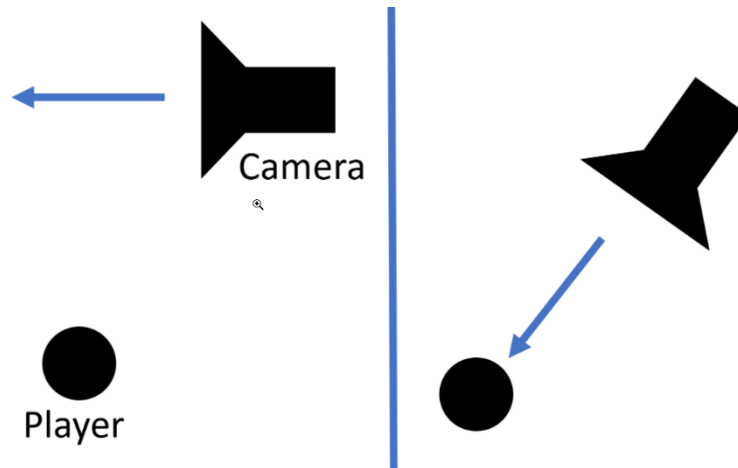


En el objeto transform podemos encontrar varios atributos los más importantes son:

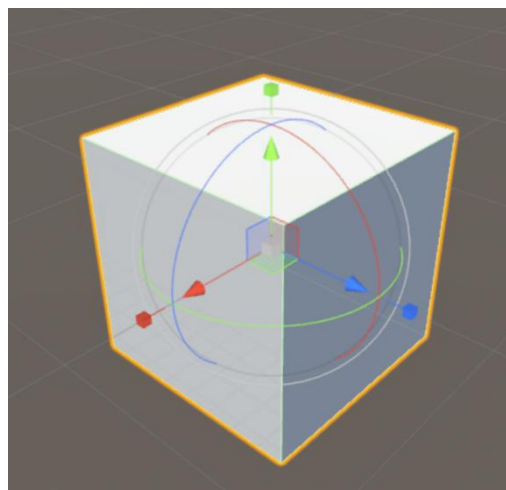
- **eulerAngles:** Devuelve un vector3 con los valores de la rotación del GameObject representados en ángulos, estos valores suelen ser distintos a los almacenados en el transform por este motivo.
Los eulerAngles no son recomendados a la hora de trabajar con rotaciones ya que pueden llegar a causar confusiones.
- **localPosition:** Posición donde se encuentra el GameObject respecto al objeto padre.
- **localScale:** Tamaño del GameObject respecto al objeto padre.
- **localRotation:** Rotación que tiene el GameObject respecto a la del objeto padre.
- **position:** Devuelve la posición del objeto en la escena.
- **rotation:** Devuelve la rotación del objeto en la escena.

Transform tiene un [gran número de métodos](#) que son muy útiles a la hora de crear un videojuego como pueden ser:

- **LookAt:** Este método se encarga de **rotar el objeto** para que mire hacia el punto que se le indique (puede ser otro objeto o coordenada), en su declaración podremos encontrar dos parámetros uno obligatorio y otro opcional, el obligatorio es de tipo Transform donde tendremos que especificar el punto donde se orientará nuestro objeto y el opcional es un vector3 que indicara la dirección del vector 'up'.



- **Rotate:** Permite la rotación del objeto, se debe **pasar obligatoriamente un Vector3** con los valores de rotación, también podremos pasarle un Space donde podremos determinar si queremos que el GameObject rote localmente o en relación con la escena.



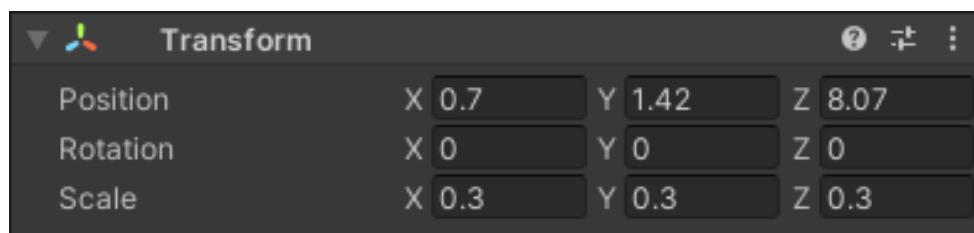
- **SetPositionAndRotation:** Con este método podremos mover y rotar el objeto a la vez tendremos que pasar por parámetro un Vector 3 referente a la posición y Quaternion para la rotación.

5.1.2 Principales Game Objects Utilizados

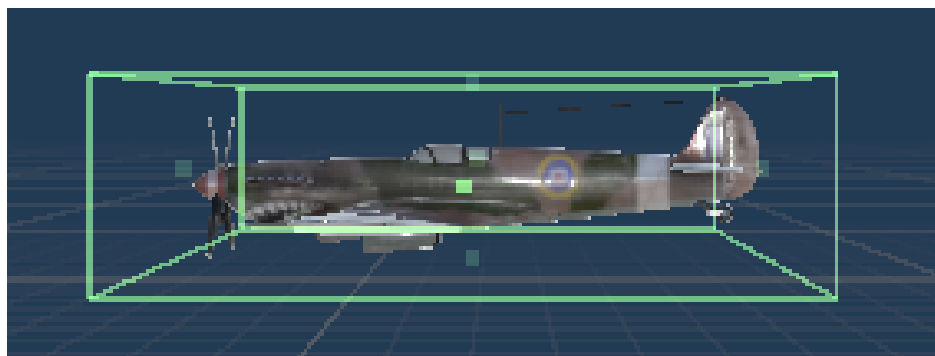
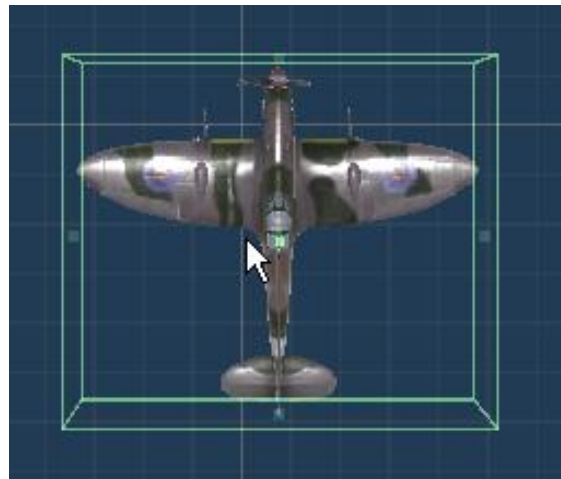
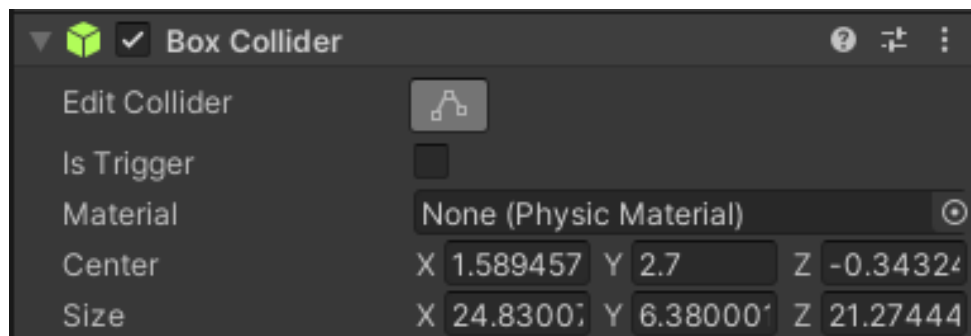
GameObject Avion:

Objeto 3D obtenido de la asset store de Unity, el nombre del complemento es **Super Spitfire** este GameObject varios componentes:

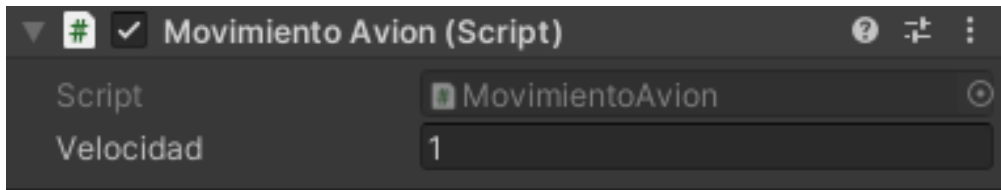
- **Transform:** Contiene **datos de escala y posición** necesarios para que el GameObject tenga unas dimensiones acordes a los demás elementos del videojuego.



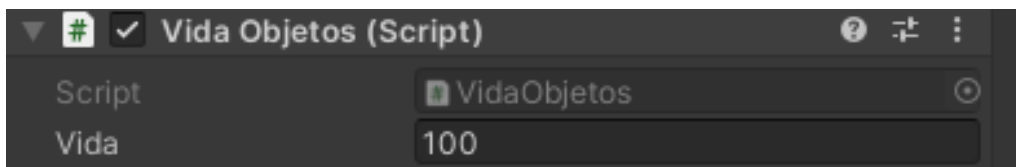
- **Box Collider:** Genera un **área alrededor de nuestro Spitfire** donde puede ser impactado con otros objetos. Esta hitbox debe ser lo más ajustada a nuestro objeto.



- **Movimiento Avión (Script):** Script que se encarga del movimiento del avión en el mapa, tiene una variable publica de tipo float llamada “velocidad” que nos permitirá modificar la velocidad del avión sin necesidad de entrar al código.



- **Vida (Script):** Script que contienen todos los objetivos del juego (dianas, tanques y aviones).
Tiene una variable publica llamada vida donde podremos ajustar la vida de nuestro avión en cualquier momento.

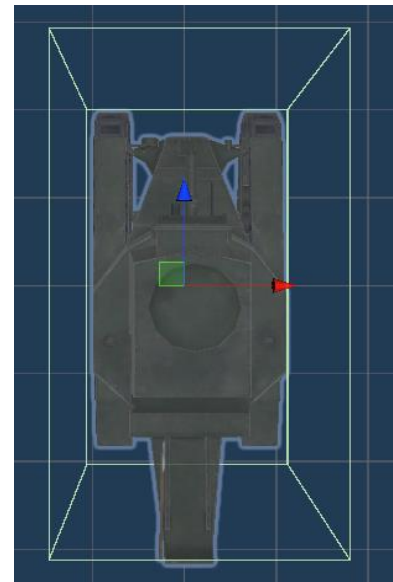
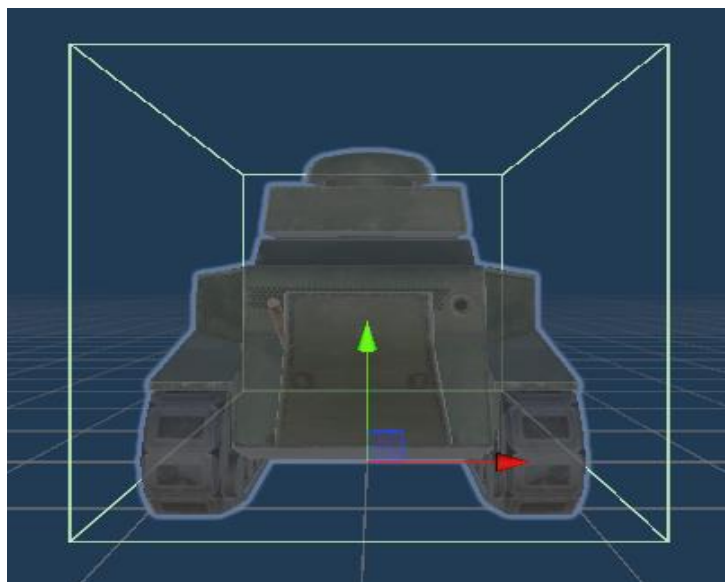
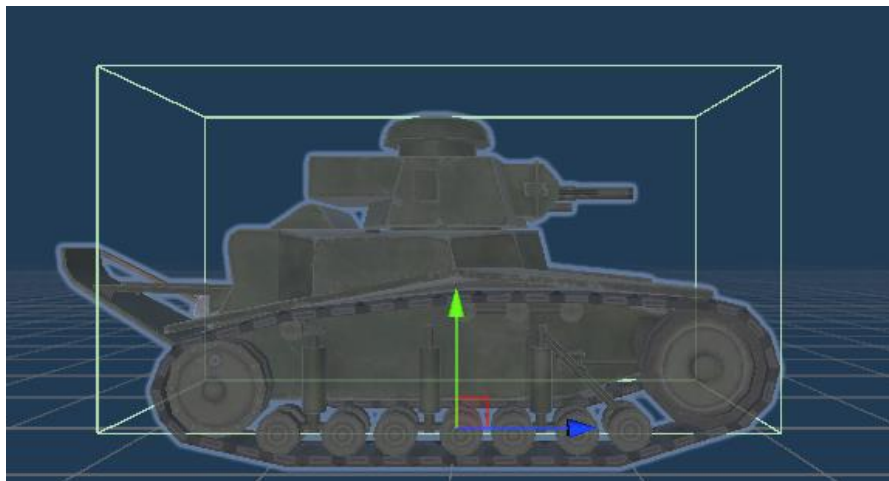
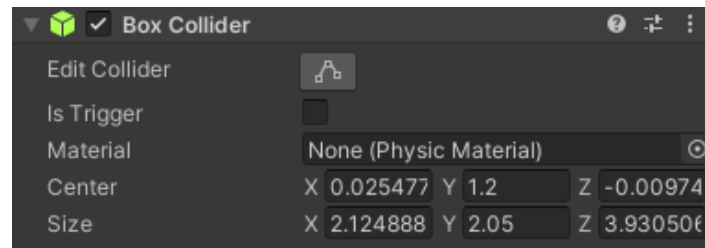


GameObject Tanque:

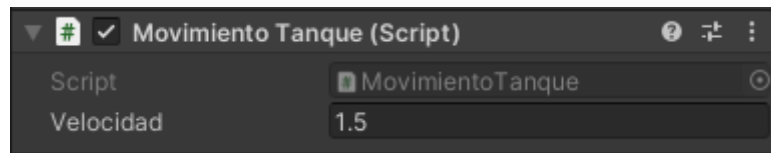
Objeto 3D obtenido de la asset store de Unity, el nombre del complemento es **MC-1 USSR Toon Tank** este GameObject varios componentes:

- **Transform:** Contiene datos de escala y posición necesarios para que el tanque tenga unas dimensiones acordes a los demás elementos del videojuego.

- **Box Collider:** Genera un área alrededor de nuestro MC-1 donde puede ser impactado con otros objetos. Esta hitbox debe ser lo más ajustada a nuestro objeto.



- **Movimiento Tanque (Script):** Script que se encarga del movimiento del tanque en el mapa, tiene una variable publica de tipo float llamada “velocidad” que nos permitirá modificar la velocidad del tanque sin necesidad de entrar al código.

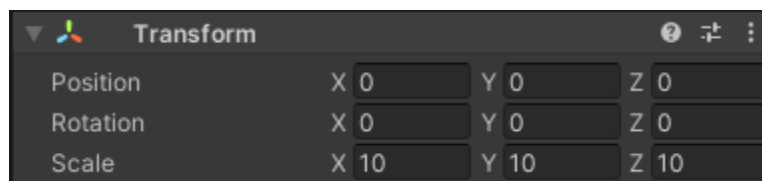


- **Vida (Script):** Script que contienen todos los objetivos del juego (dianas, tanques y aviones).
Tiene una variable publica llamada vida donde podremos ajustar la vida de nuestro tanque en cualquier momento.

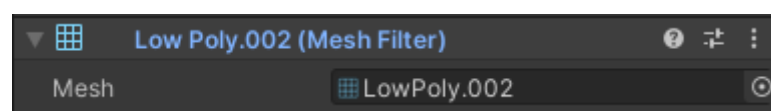
GameObject Diana:

Objeto 3D obtenido de la asset store de Unity, el nombre del complemento es **Military target** este GameObject varios componentes:

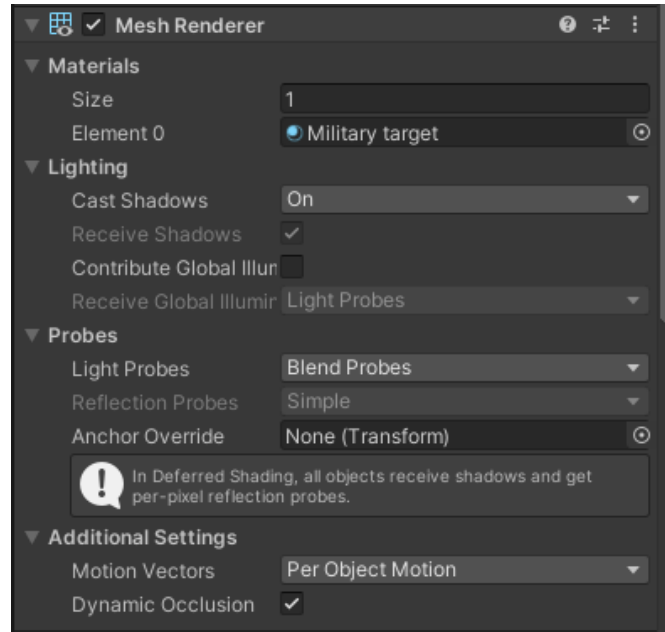
- **Transform:** Contiene datos de escala y posición necesarios para que la diana tenga unas dimensiones acordes a los demás elementos del videojuego.



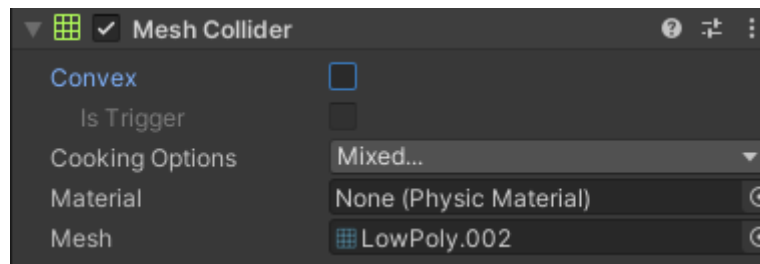
- **Mesh Filter:** Se encarga de tomar un mesh de alguno de nuestros assets y se lo pasa al mesh render para que sea renderizado por pantalla.

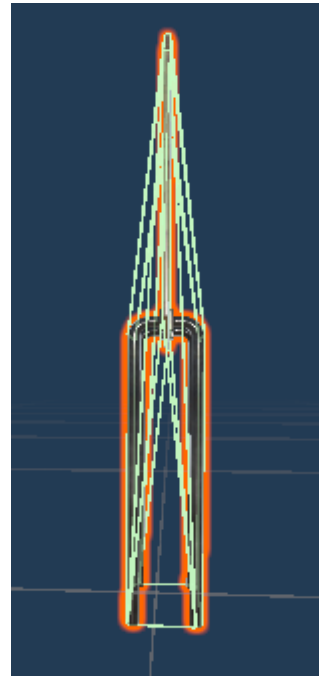
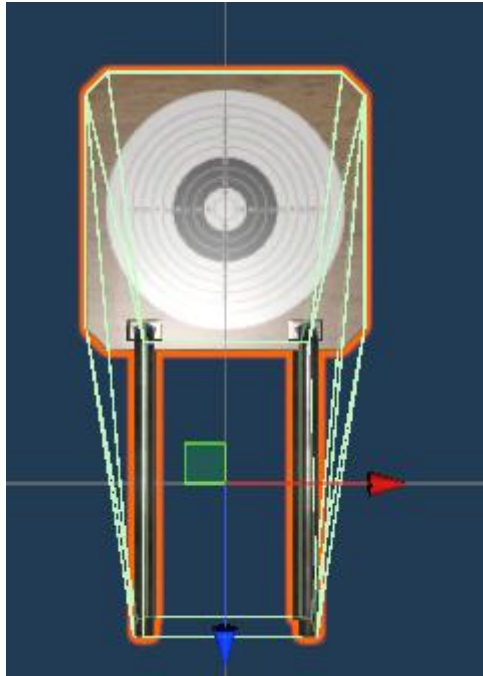


- **Mesh Renderer:** Componente que toma la geometría del Mesh Filter y la renderiza en la posición definida por el componente Transform del GameObject.



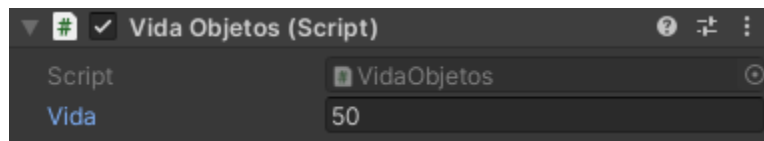
- **Mesh Collider:** Genera una hitbox alrededor del GameObject, lo consigue gracias al Mesh de nuestro objeto.





- **Vida (Script):** Script que contienen [todos los objetivos del juego](#) (dianas, tanques y aviones).

Tiene una variable publica llamada vida donde podremos ajustar la vida de nuestro avión en cualquier momento.



5.1.3 Principales Scripts del juego

ControlArma(Script)

Clase base de nuestro juego que se encarga tanto de la [creación de los objetos](#) (tanques, dianas y aviones), [generaciones de las explosiones](#) de salida y de impacto de la bala, realizar el conteo de los puntos obtenidos y [destrucción de los objetos](#) cuando su vida sea cero.

En cuanto a las variables las podemos clasificar en los grupos:

- **Objetos utilizados:** Gameobjects que se utilizan en la clase, en las variables públicas “explosionimpacto” y “efectodisparo” deberemos añadir las animaciones de explosiones que se usan a la hora de disparar el cañón y el impacto de la bala, en las variables públicas “avión”, “diana” y “tanque” añadiremos los modelos prefabricados de cada uno, y por último la variable “ObjetoImpactado” que guardará el objeto que el raycast golpee.

```
public GameObject explosionimpacto;  
public GameObject efectodisparo;  
public GameObject diana;  
public GameObject tanque;  
public GameObject Avion;  
GameObject ObjetoImpactado;
```

- **Variables generales:** Estas variables son de tipos variados que se necesitan para el correcto funcionamiento del videojuego.

Por un lado, las variables públicas:

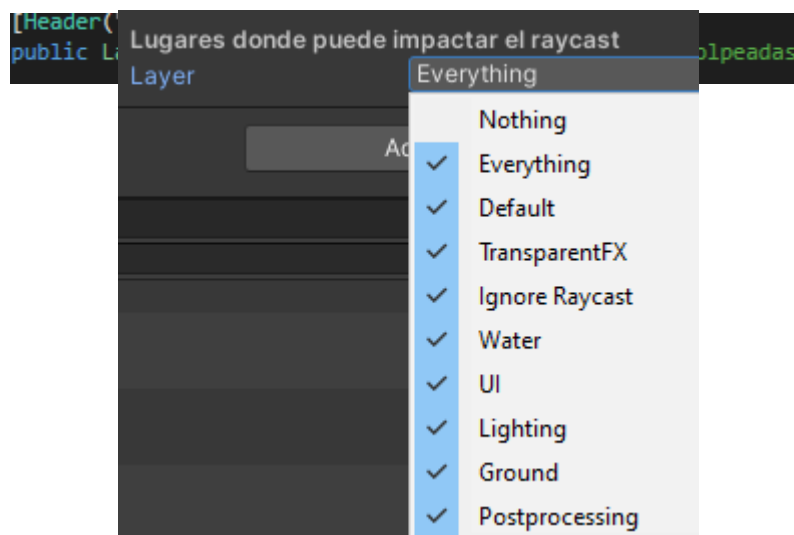
- “**boquilla**” trasnfrom de la boquilla del cañón.
- “**rangodediparo**” hasta dónde puede llegar el disparo.
- “**mayorX**,” “**menorX**,” “**mayorZ**” y “**menorZ**” se encargarán de limitar hasta dónde pueden generarse las posiciones aleatorias de nuestras dianas.
- “**daño**” daño que le produce a los objetivos nuestro tanque.
- “**numdianas**”, “**numaviones**” y “**numtanques**” numero objetos creados de cada tipo.
- “**TiempoRecarga**” tiempo que debe esperarse entre disparo y disparo.

Las variables privadas que encontramos son:

- “**sigdisparo**” se encarga de regular la espera entre disparos.
- “**camarajugadortransfrom**” variable tipo transform que almacenará el transform de la cámara.
- “**puntuación**” puntuación general que lleva el jugador (cada vez que empiece el juego se pondrá a 0).

```
[Header("Variables generales")]
public Transform boquilla;
public float rangodedisparo;
public float mayorX;
public float menorX;
public float mayorZ;
public float menorZ;
public int daño;
public int numtanque;
public int numdianas;
public int numaviones;
public float TiempoRecarga;
private float sigdisparo;
private Transform camarajugadortransform;
private int puntuacion = 0;
```

- **Lugares donde puede impactar el raycast:** Esta categoría consta únicamente de una variable pública tipo [LayerMask](#) llamada “layer” en la cual podremos elegir que objetos pueden ser impactados por nuestro disparo.



En el [método Start\(\)](#) que se ejecutará cada vez que se ejecute el juego se guardará en la variable “[camarajugadortransform](#)” la cámara del jugador, esto se consigue gracias al método [FindGameObjectWithTag\(\)](#) donde pondremos el tag que tiene nuestra cámara, aparte de esto se llamarán a los métodos encargados de la creación de nuestras dianas ([creacionDianas\(\)](#)), tanques ([creacionTanque\(\)](#)) y aviones ([creacionAvion\(\)](#)).

```
private void Start()
{
    camarajugadortransform = GameObject.FindGameObjectWithTag("MainCamera").transform;
    creacionDianas();
    creacionTanque();
    creacionAvion();
}
```

Los métodos de creación de objetos son muy similares entre si la única diferencia es la posición y la rotación inicial que tienen.

En estos métodos tendremos tres variables “**cont**” que se utilizará como contador para controlar el número de clones que se crean de cada objeto, “**xpos**” y “**zpos**” que guardarán las posiciones donde se va generar nuestros objetivos.

Una vez definidas estas variables **comenzará un bucle** que no terminará hasta que la variable “**cont**” sea igual al número de objetos que queremos crear, en este bucle empezará con la generación de dos números aleatorios que servirán de coordenadas para la creación de los objetos, estos números serán tomados entre las variables “**mayorX**,” **menorX**,” **mayorZ**” y “**menorZ**” definidas anteriormente.

Tras esto se generará el clon del objeto con el método **Instantiate()** le pasaremos por parámetro el objeto que queremos clonar, su posición inicial y su rotación, la rotación en la coordenada Y la contraria a la cámara para que este de frente y en la z la misma que la de la cámara (en el caso de las dianas la rotación de X será igual a noventa para que se encuentre de pie).

creacionDianas()

```
private void creacionDianas()
{
    int cont = 0;
    float xpos;
    float zpos;
    while (cont < numdianas)
    {
        xpos = Random.Range(mayorX, menorX);
        zpos = Random.Range(mayorZ, menorZ);

        Instantiate(diana, new Vector3(xpos, 2f, zpos), Quaternion.Euler(90f, -camarajugadortransform.eulerAngles.y, camarajugadortransform.eulerAngles.z));
        cont++;
    }
}
```

creacionTanque()

```
private void creacionTanque()
{
    int cont = 0;
    float xpos;
    float zpos;

    while (cont < numtanque)
    {
        xpos = Random.Range(mayorX, menorX);
        zpos = Random.Range(mayorZ, menorZ);

        Instantiate(tanque, new Vector3(xpos, 2f, zpos), Quaternion.Euler(0, -camarajugadortransform.eulerAngles.y, camarajugadortransform.eulerAngles.z));
        cont++;
    }
}
```

creacionAvion()

```
private void creacionAvion()
{
    int cont = 0;
    float xpos;
    float zpos;
    float ypos;

    while (cont < numaviones)
    {
        xpos = Random.Range(mayorX, menorX);
        zpos = Random.Range(mayorZ, menorZ);
        ypos = Random.Range(10f, 20f);

        Instantiate(Avion, new Vector3(xpos, ypos, zpos), Quaternion.Euler(0, -camarajugadortransform.eulerAngles.y, camarajugadortransform.eulerAngles.z));
        cont++;
    }
}
```

Una vez generados todos los objetivos comenzara la ejecución del método *update()*, que llamará al método *disparos()* que se encargará de todo lo referente a los disparos, restar vida y sumar los puntos.

Una vez comience la ejecución del método *disparos()*, encontraremos una condición cuyo código se ejecutará en el caso de que se pulse el botón que activa nuestro disparo y haya pasado el tiempo especificado para hacer el siguiente disparo.

```
if (Input.GetButtonDown("Fire1") && Time.time > sigdisparo)
```

Una vez entramos en el `if` inicializamos la variable “golpe” de tipo `RaycastHit`, crearemos un `GameObject` llamada “explosionsalida” donde guardaremos un clon del efecto de explosión del cañón que se generará en la boquilla, una vez generada la destruiremos tras pasar un segundo.

Una vez se generó la explosión generaremos un `raycast` que nos devolverá un booleano que será verdadero en el caso de que el disparo impacte con algún objeto (que hayamos puesto en nuestra variable “layer” que pueda ser golpeado), los datos del disparo serán guardados en nuestra variable “golpe”.

Si el `raycast` ha golpeado algún objeto crearemos una explosión en el punto exacto donde impacte, el cual sabemos gracias a nuestra variable tipo `RaycastHit`.

El objeto golpeado será guardado en nuestra variable “ObjetoImpactado” (creada al inicio de la clase), a este objeto le restaremos a su vida el daño de nuestro disparo, esto lo conseguimos gracias al método creado en la clase “VidaObjetos” que tienen todos los objetos añadidos, este método devuelve un número entero que nos informa de la vida restante del `GameObject`.

En el caso de que esta vida sea igual o menor a cero se identificara el tag del objeto y el número de puntos que da cada uno(en el método “sumarPuntos”).

Posteriormente se eliminará el objeto.

```
private int sumarPuntos(GameObject objetoasumar)
{
    if (objetoasumar.tag == "Tanque")
    {
        return 100;
    }
    else if (objetoasumar.tag == "Avion")
    {
        return 200;
    }
    else if (objetoasumar.tag == "Diana")
    {
        return 50;
    }
    return 0;
}
```

```

private void disparos()
{
    if (Input.GetButtonDown("Fire1") && Time.time > sigdisparo)
    {
        sigdisparo = Time.time + TiempoRecarga;
        RaycastHit golpe;

        GameObject explosionsalida = Instantiate(efectodedisparo, boquilla.position, Quaternion.Euler(boquilla.forward));

        Destroy(explosionsalida, 1);

        if (Physics.Raycast(camarajugadortransform.position, camarajugadortransform.forward, out golpe, rangodedisparo, layer))
        {
            GameObject agujerodebalaimagencione = Instantiate(explosionimpacto, golpe.point + golpe.normal, Quaternion.LookRotation(golpe.normal));
            Destroy(agujerodebalaimagencione, 2);
            ObjetoImpactado = golpe.transform.gameObject;
            int vida_restante = ObjetoImpactado.GetComponent<VidaObjetos>().restar_vida(1);

            if (vida_restante <= 0)
            {
                puntuacion = puntuacion + sumarPuntos(ObjetoImpactado);
                Destroy(ObjetoImpactado);
            }
        }
    }
}

```

Movimiento del avión/tanque (Script):

Script que se encarga del movimiento del avión y del tanque por la escena.

En la cabecera de la clase hemos creado 3 variables una pública llamada “**velocidad**” de tipo **float** que se encarga de regular la velocidad de nuestros vehículos y las dos son de tipo **Vector3**, la llamada “Destino” se encargará de almacenar el punto donde se dirigirá nuestro avión o tanque y la denominada “**Dirección**” se utilizará para guardar la diferencia entre la posición actual del objeto y la posición de destino.

Cuando el objeto se genere se ejecutará el método **Start()** que a su vez llamara al método **generarLugaredestino()**.

```

public float velocidad;
Vector3 Destino;
Vector3 Direccion;

```

```

void Start()
{
    generarlugaredestino();
}

```

```

private void generarlugaredestino()
{
    float radioesfera = 100f;
    //Generamos una posicioon de destino, sumando a nuestra poscion actual un punto aleatorio de una esfera generada a nuestro alrededor con radio x
    Destino = transform.position + Random.insideUnitSphere * radioesfera;
}

```

Este método generará un lugar de destino para el avión y el tanque, esto se consigue gracias a la función “Random.insideUnitSphere” que generará una esfera de radio personalizable alrededor del objeto y un punto de esa figura geométrica será elegido aleatoriamente como punto de destino.

En el caso del avión podría darse el caso de que genere una posición de destino por debajo del suelo esto se solucionará cambiando la coordenada Y por 5, así eliminaríamos el problema de que el avión se vaya del plano.

```
if (Destino.y < 1) {  
    Destino.y = 5;  
}
```

Por su parte el tanque necesitará que la posición de la Y sea constante para que se quede en el suelo, para conseguir esto tras generar el número aleatorio se cambiara el valor de la Y para que no se levante del suelo.

```
Destino = transform.position + Random.insideUnitSphere * radioesfera;  
Destino.y = 1.2f;  
transform.LookAt(Destino);
```

Por último, a la hora de generar el punto destino es hacer que nuestros objetos miren hacia donde tiene que moverse, gracias al método LookAt() de la clase transform conseguimos que los vehículos miren hacia el destino y avance de frente.

```
transform.LookAt(Destino);
```

Una vez generada la posición de destino y hecha la recalibración de nuestro Spitfire, cada vez que se cambie de fotograma se ejecutará el método update().

Al comenzar la ejecución del método "update" guardaremos en la variable dirección la diferencia entre la posición actual y la de destino, en el caso de que esta sea igual a cero, se invocará de nuevo al método `generarlugardedestino()` para generar una nueva posición, si no, el avión se ira moviendo de forma continua hasta el punto destino (con `Time.deltaTime` conseguimos que el movimiento sea uniforme en todos los ordenadores donde se ejecute el video juego y no dependa de los componentes hardware).

```
void Update()
{
    Direccion = Destino - transform.position;
    if (transform.position == Destino)
    {
        generarlugardedestino();
    }
    transform.position += Direccion.normalized * velocidad * Time.deltaTime;
}
```

Vida(Script)

Script que contiene la vida del objetivo y se encarga de la disminución de esta cuando es impactado por una bala.

En esta clase podremos encontrar una variable publica llamada "vida", la cual podremos retocar en cualquiera de los objetos donde se encuentre el Script independientemente de los valores que tenga en los otros.

```
public class VidaObjetos : MonoBehaviour
{
    public int Vida;
```

En esta clase el método `Start()` y el `Update()` estarán vacíos, únicamente encontraremos un método que devolverá un número entero llamado `restar_vida`, al cual se le deberá pasar un atributo “int” que será el daño que hace nuestro disparo.

En este método como su propio nombre indica se encarga de restar la vida que le resta al objeto el daño producido por el impacto, se devuelve la vida restante para poder eliminarlo en el caso de que sea cero.

```
public int restar_vida(int daño)
{
    Vida = Vida - daño;
    return Vida;
}
```

Script de movimiento del jugador:

Hacer el movimiento del jugador hubiera sido sencillo en un videojuego convencional, pero en nuestro caso se complicó un poco ya que se tenía que mover la maqueta al mismo tiempo que el jugador en el videojuego.

Decidimos usar un mando de `Xbox` para los controles debido a que sería más sencilla la integración con el tipo de juego que tenemos.

Unity recientemente integró un sistema de input mucho más sencillo que los anteriores, el [new Input System](#).

Este sistema es capaz de detectar pulsaciones de los botones de cualquier controlador que se conecte, tanto sus [joysticks](#) como sus gatillos.

Input System
Version 1.0.2 - January 21, 2021 2019.4 verified
Name
com.unity.inputsystem
Links
[View documentation](#)
[View changelog](#)
[View licenses](#)
Author
Unity Technologies
Registry Unity
Published Date
January 21, 2021

A new input system which can be used as a more extensible and customizable alternative to Unity's classic input system in UnityEngine.Input.

En nuestro caso vamos a detectar el movimiento de los joysticks izquierdo y derecho. Para ello, creamos una variable [Gamepad](#), a la que le añadimos el tipo de dispositivo que queremos capturar.

```
Gamepad gp;  
gp = InputSystem.GetDevice<Gamepad>();
```

Para leer los valores de los joysticks debemos llamar a los métodos de cada joystick “[leftStick](#)” para la izquierda y “[rightStick](#)” para la derecha, además de especificar que eje queremos leer. Como resultado nos devolverá un valor tipo float con un número de entre 1 y 0.

```
moveX = gp.leftStick.ReadValue().x;  
moveY = gp.leftStick.ReadValue().y;  
cañon = gp.rightStick.ReadValue().y;  
torreta = gp.rightStick.ReadValue().x;
```

Para realizar el movimiento de un jugador bastaría con hacer que avance conforme a un vector3 al que le pasamos los valores de un joystick.

```
player.transform.Translate(new Vector3(moveX, 0, moveY));
```

Pero como nosotros además debemos determinar el movimiento del tanque enviándole unas órdenes específicas por cada dirección, por lo que con varios “if” comprobamos que posición tiene el joystick en cada posición.

Luego, enviamos los comandos a la placa [NodeMCU](#), y por último aplicamos diferentes fuerzas a un [rigidBody](#), que está sujeto al motor de físicas de Unity, para hacer un movimiento más natural.

```
//Condición para determinar la dirección del joystick
if ((moveX > 0.4) && (moveY < 0.4) && (moveY > -0.4)){

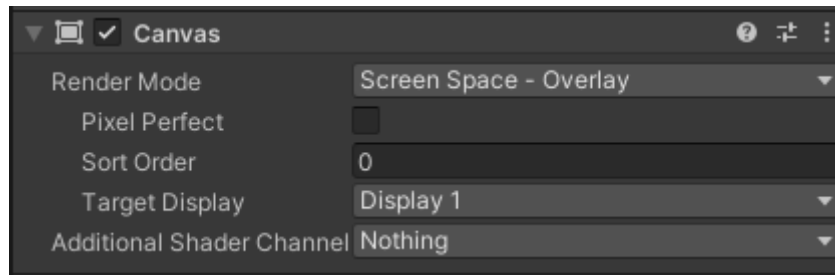
    //Código para mandar a Arduino
    UduinoManager.Instance.digitalWrite(16, 255);
    UduinoManager.Instance.digitalWrite(5, 0);
    UduinoManager.Instance.digitalWrite(13, 255);
    UduinoManager.Instance.digitalWrite(15, 0);

    //Movimiento en Unity
    rb.AddForce(posCamara * force/10, ForceMode.Acceleration);
    rb.AddTorque(new Vector3(0, 90, 0));
}
```

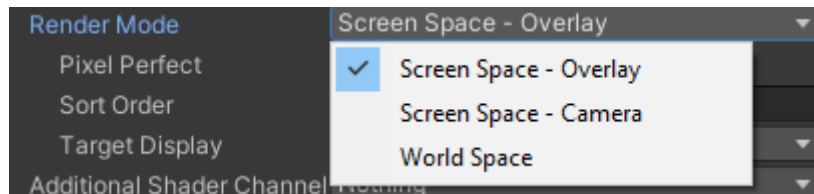
HUD (Head-up Display):

Para la creación de nuestro HUD (información que en todo momento se muestra en pantalla durante la partida), para su elaboración hemos utilizado un objeto de unity llamado [Canvas](#) (Área donde se encuentran todos los elementos de una interfaz de usuario), podemos encontrar los siguientes componentes:

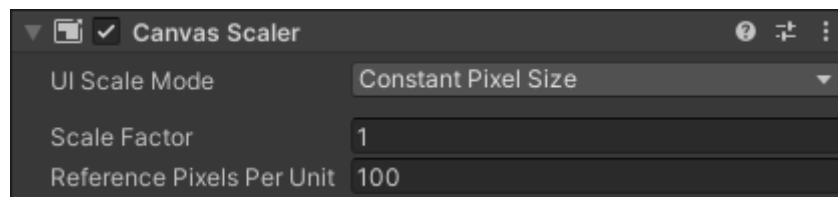
- **Canvas:** Espacio abstracto donde la interfaz de usuario se coloca y se renderiza.



Además, podemos elegir de qué manera queremos que sea renderizado nuestro UI y con la propiedad Target Display podemos configurar la pantalla de destino de nuestro canvas.



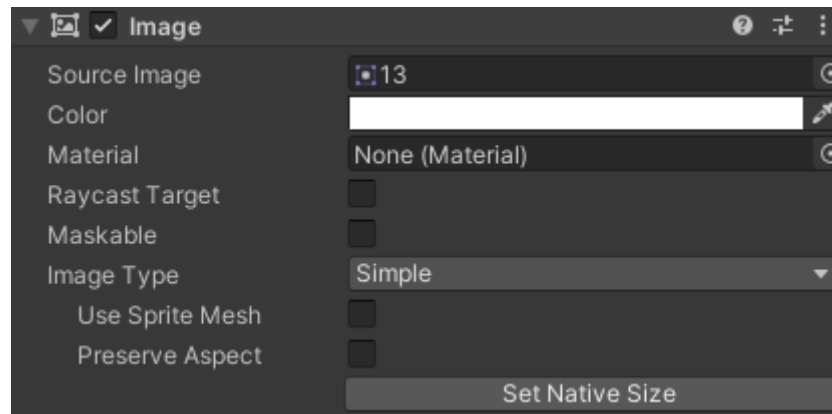
- **Canvas Scaler:** Es utilizado para controlar la escala de nuestra interfaz de usuario y la densidad de pixeles, estos cambios afectan a los demás componentes del canvas que se encuentren por debajo.



Dentro de nuestro canvas añadiremos como elementos hijos la cruceta (para saber a qué lugar va nuestro disparo) y la barra de recarga (una forma más visual de saber cuándo se puede volver a disparar).



Para la cruceta hemos añadido un objeto vacío, al cual hemos añadido un componente image.



En el apartado [Source Image](#) de este componente deberemos añadir nuestro diseño de mira, gracias a la propiedad color podemos cambiar el color de nuestra cruceta de una manera fácil.



En cuanto a la barra de recarga, se trata de un [objeto vacío](#) al cual hemos añadido un componente [image](#) y un script llamado “[ReloadBar](#)”, este objeto definirá el cuadro vacío donde se va a situar nuestra barra de recarga y se encargará de controlarla.

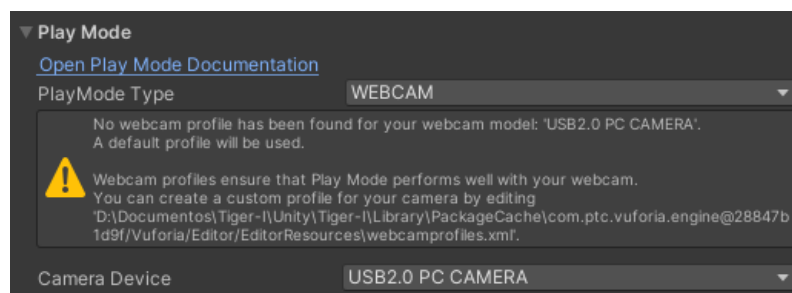
Vuforia:

Como hemos visto antes, Vuforia tiene una buena compatibilidad con Unity además de ser bastante sencilla su instalación. Ahora vamos a configurarlo para hacer que, cuando ejecutemos el juego, el escenario sea la imagen que recibe Vuforia.

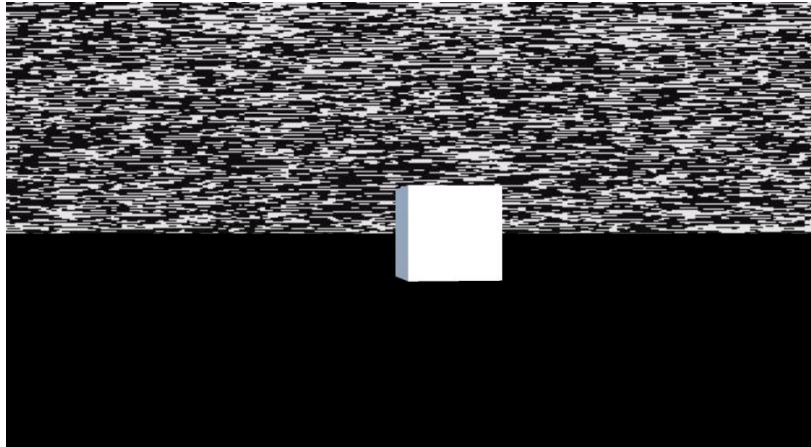
Primero debemos ir a la configuración de Vuforia, haciendo click en el botón del script “Vuforia Behaviour”.



Una vez dentro debemos seleccionar que el modo de inicio sea webcam y el dispositivo que va a capturar la imagen sea el indicado, en nuestro caso , se llama USB2.0 PC CAMERA.

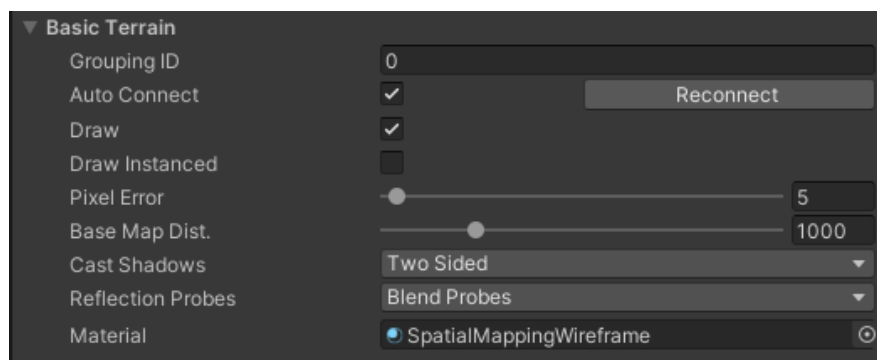


Una vez que tengamos game object AR Camera como hemos visto anteriormente, el juego se vería así.(Las interferencias se deben a que el emisor de video no está conectado)

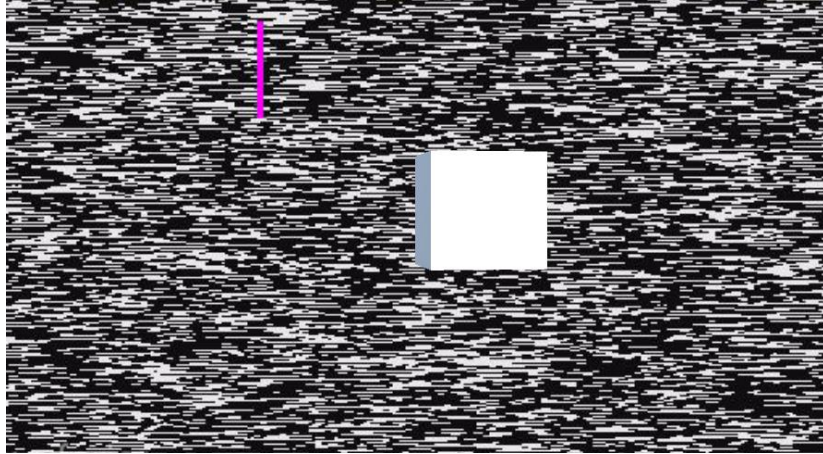


Para dar la sensación de realidad aumentada real, el suelo, debemos hacer que sea invisible (Es necesario que exista, ya que con las físicas de Unity, los objetos caerían al infinito).

Para ello vamos al objeto terrain y en la configuración deshabilitamos la opción de dibujado “draw”.



De esta forma cuando la cámara capte un escenario, el cubo parece que está en el suelo.

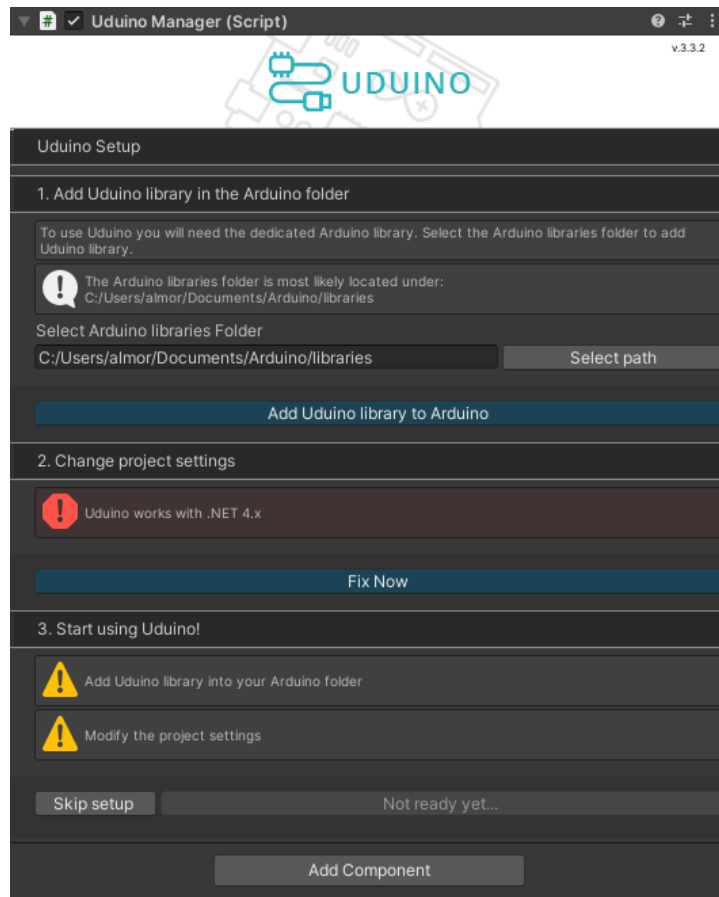


Por el momento, nuestro proyecto solo va a poder ser ejecutado dentro del entorno de Unity, ya que, estamos esperando a que el equipo de desarrollo de Vuforia, publique la actualización para el soporte de Windows, de momento sólo tiene soporte con Android.

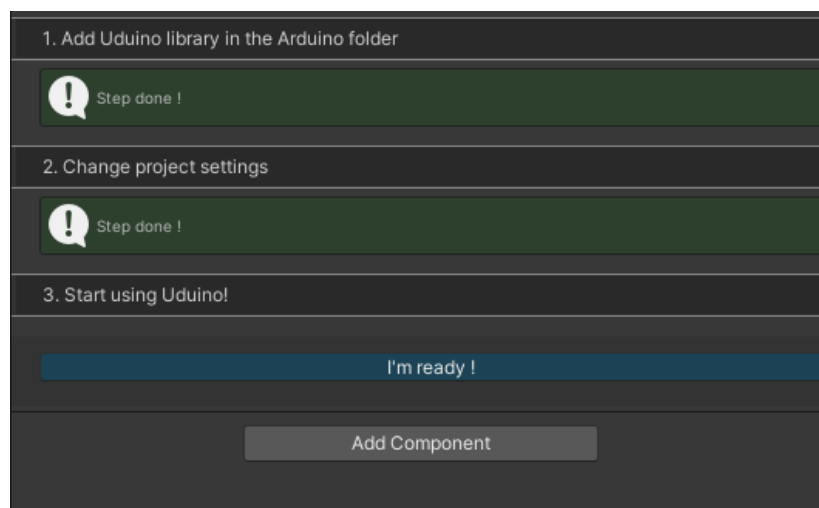
Plugin Uduino

Como ya hemos visto, el plugin de Uduino nos permite comunicar nuestras placas de Arduino con Unity de una forma muy sencilla, por ello, vamos a configurarlo.

Una vez que el plugin esté instalado, arrastraremos un objeto Uduino al árbol de objetos.

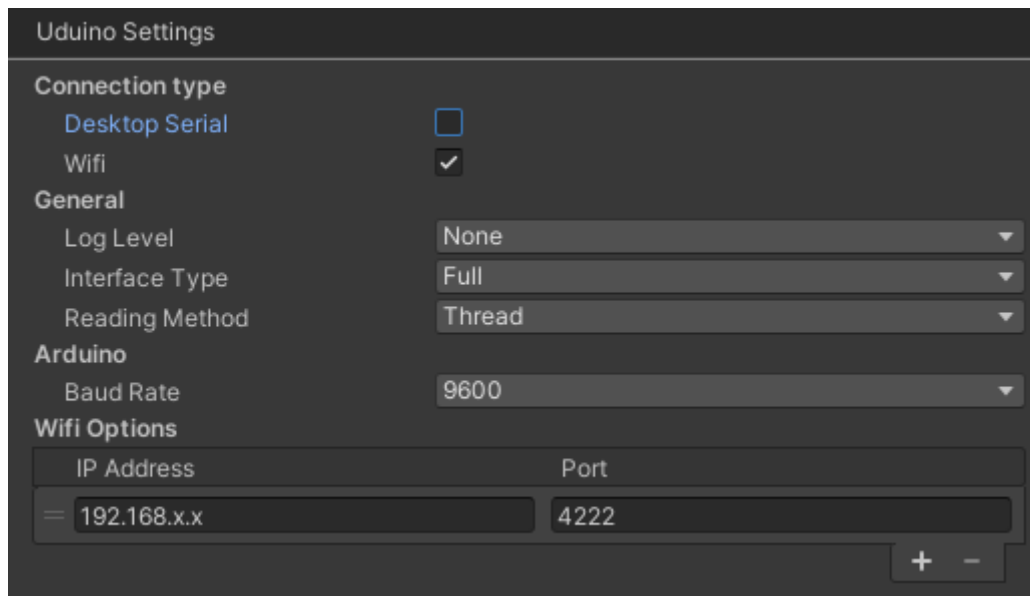


Como podemos ver en la imagen Uduino ofrece un script en el que podemos configurar los pasos iniciales sin tocar ni una línea de código. Primero debemos seleccionar la ubicación de nuestra librería de Arduino (este debe estar previamente instalado), después, hay que cambiar la versión de .NET ya que Uduino funciona con la version 4 (por defecto, Unity está en 3.5).

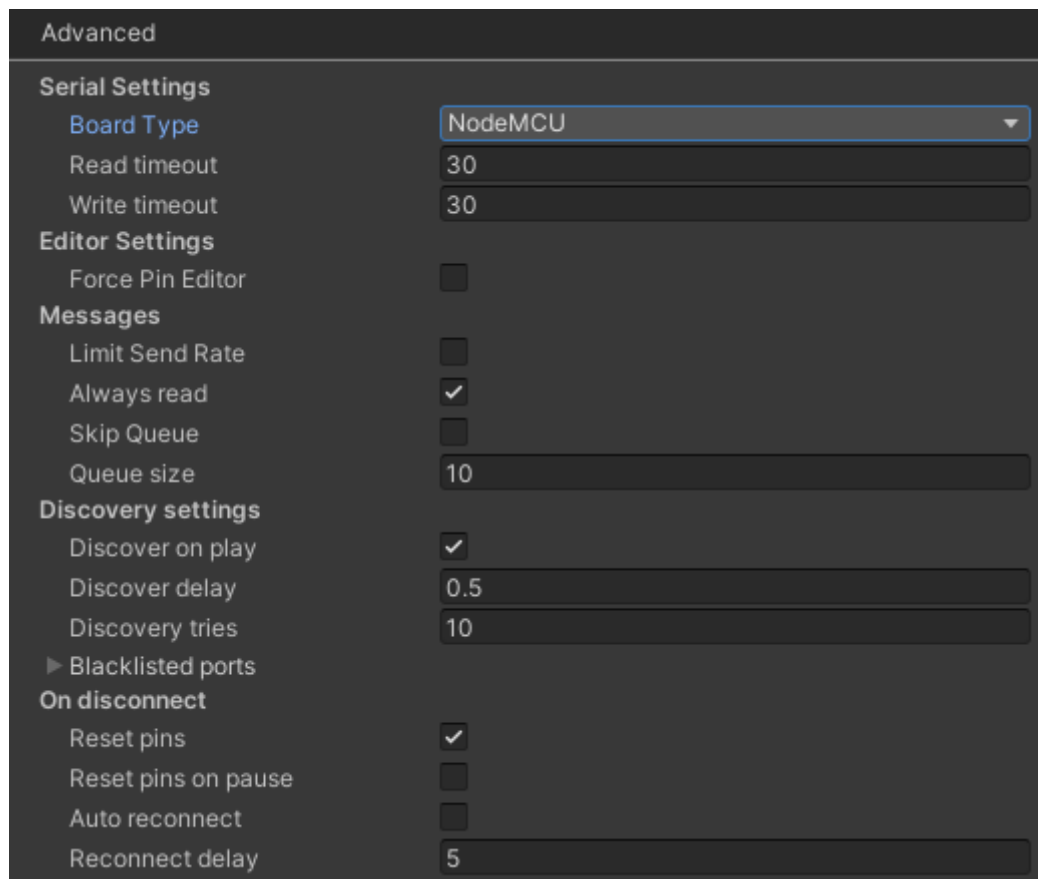


Cuando hayamos terminado, daremos al botón de listo, para configurar la conexión de nuestra placa.

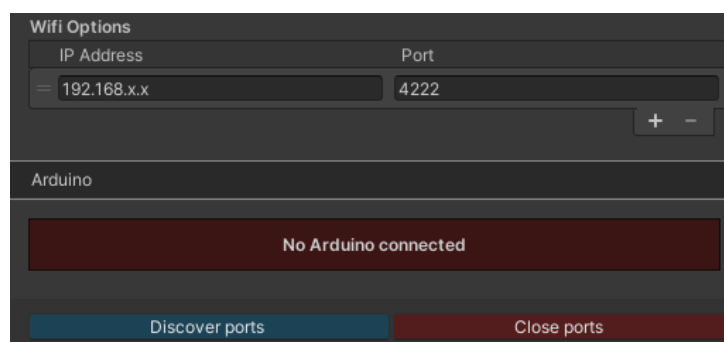
Primero vamos a configurar el método de conexión, como hemos visto, nuestro proyecto va a usar wifi para la conexión, por lo que vamos a activar la casilla de wifi y deseleccionar la casilla de la comunicación serial (comunicación por cable).



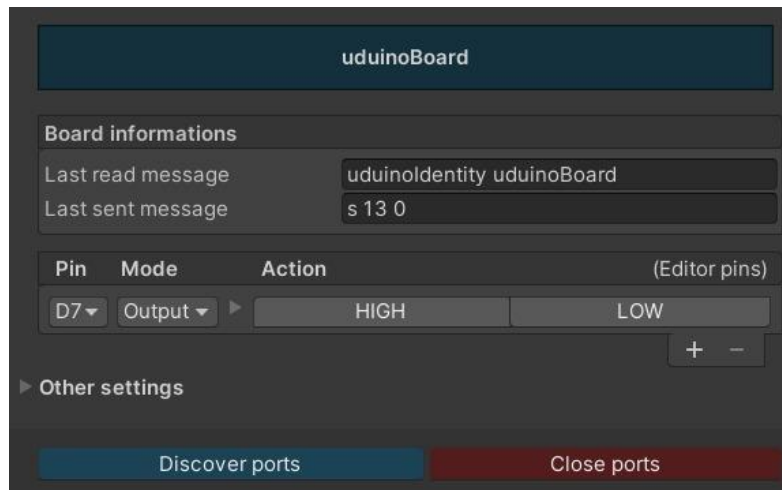
Otra cosa que debemos configurar aquí es el baud rate, que debe ser el mismo que haya en la placa de wifi para que se puedan comunicar, por defecto será 9600. (Los baudios representan el número de unidades de señal por segundo que se van a transmitir, por lo que, si uno emite a 115200 y otro lee a 9600 no van a poder entenderse entre sí);



En las opciones avanzadas el único parámetro que debemos cambiar es el tipo de placa, nosotros vamos a conectarnos a una placa NodeMCU, aunque hay otras opciones interesantes para configurar como el delay en la lectura o escritura, que hacer cuando se inicia la búsqueda o cuando se desconecta, etc.



Para establecer la conexión debemos conectar tanto Arduino como el ordenador a una misma red wifi. A continuación, indicamos la ip y el puerto al que la placa está conectada (más tarde veremos cómo se obtiene), y posteriormente pulsaremos al botón “discover ports”. Si hemos hecho todo correctamente debería conectarse.



Cuando esté conectada vamos a ver este cuadro, desde aquí podemos enviar mensajes y comandos a nuestra placa, aunque para enviar mensajes mientras se ejecuta el juego, hay que hacerlo desde un script.

```
UduinoManager.Instance.pinMode(16, PinMode.Output);
UduinoManager.Instance.pinMode(5, PinMode.Output);
UduinoManager.Instance.pinMode(13, PinMode.Output);
UduinoManager.Instance.pinMode(15, PinMode.Output);
```

Este es un ejemplo de cómo usamos la clase de Uduino, el método pinMode inicializa un pin y requiere de dos parámetros, el código del pin de la placa NodeMCU y el modo, Output para escribir e Input para leer.

Nosotros inicializamos 4 pines que serán los que se comunicarán con la placa de Arduino.

```
UduinoManager.Instance.digitalWrite(16, 255);  
UduinoManager.Instance.digitalWrite(5, 0);  
UduinoManager.Instance.digitalWrite(13, 255);  
UduinoManager.Instance.digitalWrite(15, 0);
```

Este otro ejemplo lo que hace es escribir en los pines. Como antes el primer parámetro es el código del pin, pero el segundo es la intensidad del pulso.

También podemos escribir y leer los pines en modo analógico con “analogWrite”.


Hay muchísimas más funciones que no hemos necesitado y que se pueden ver en la ayuda que Uduino nos proporciona en [este enlace](#).

5.2 Arduino.

A continuación, vamos a explicar el desarrollo de las partes más importantes del proyecto en Arduino, tanto de los motores como de la conexión con Unity.

NodeMcu:

Para poder programar nuestra placa de wifi primero debemos instalar los drivers de [este enlace](#), después debemos configurar nuestro [IDE Arduino](#) para que reconozca este tipo de placas wifi. Para ello debemos ir a la configuración del IDE y en el apartado de “gestor de URL adicionales de tarjetas” escribiremos el siguiente enlace:

Gestor de URLs Adicionales de Tarjetas: 

Una vez configurado, en el menú de herramientas podemos usar las placas wifi que aparecen, la nuestra se llama [NodeMCU 1.0](#).

Respecto al código de esta placa, como ya hemos mencionado antes, se limita a recibir parámetros de Unity y a enviarlos a otra placa de Arduino en forma de bits.

Ahora vamos a ver algunas partes importantes del código, en primer lugar, instanciamos un objeto tipo [Uduino_Wifi](#), esta es una clase que se importa automáticamente a la librería de Arduino cuando configuramos Uduino en Unity.

```
Uduino_Wifi uduino("uduinoBoard");  
uduino.connectWifi("UduinoWIFI", "abcdefgh");
```

Después, vamos a conectarnos a nuestra red con el método `connectWifi`, como parámetro hay que pasarle el SSID de la red y la contraseña.

```
uduino.addCommand("s", SetMode);  
uduino.addCommand("d", WritePinDigital);
```

El método “`addCommand`” lo que hace es que cuando recibe por wifi el código del primer parámetro, ejecuta el método que hay en el segundo parámetro. De forma que podemos determinar las ordenes que queramos.

El método `setMode` recoge los argumentos recibidos por wifi para inicializar un pin en modo lectura o escritura.

```
void SetMode() {  
  
    int pinToMap;  
    char arg;  
  
    arg = uduino.next(); //argumento recibido por wifi  
  
    if (arg != NULL)  
    {  
        pinToMap = atoi(arg); //convierte el string del argumento en int  
    }  
  
    int type;  
  
    arg = uduino.next(); //argumento recibido por wifi  
    if (arg != NULL)  
    {  
        type = atoi(arg); //convierte el string del argumento en int  
        PinSetMode(pinToMap, type); //inicializa el pin  
    }  
}
```

Por otro lado, el método “`WritePinDigital`” escribe el pin que llegue por parámetro con la intensidad que llegue en el segundo argumento (el pin debe estar inicializado de antes).

```

void WritePinDigital() {

    int pinToMap;
    char *arg;

    arg = uduino.next(); //argumento recibido por wifi
    if (arg != NULL)
        pinToMap = atoi(arg); //convierte el string del argumento en int

    int writeValue;
    arg = uduino.next(); //argumento recibido por wifi

    if (arg != NULL && pinToMap != -1)
    {
        writeValue = atoi(arg); //convierte el string del argumento en int
        digitalWrite(pinToMap, writeValue); //escribe en el pin indicado
        Serial.println(String(pinToMap) + "    " + String(writeValue));
    }
}

```

Mega 2650

En este apartado vamos a ver el código que se instala en la placa de Arduino Mega 2650. Este [código recibe los valores binarios de 4 cables](#), según el valor recibido realizará una función u otra.

En el [loop](#) principal lo que hacemos en cada iteración es leer el valor de los cuatro cables para después, con varios “if”, determinar qué acción vamos a realizar.

```

void loop() {

    // ----- Read Data ----- //

    int p1 = digitalRead(pin1);
    int p2 = digitalRead(pin2);
    int p3 = digitalRead(pin3);
    int p4 = digitalRead(pin4);

    if(p1 != 1){

        if((p2 == 0) && (p3 == 0) && (p4 == 0)){

            stopMotors(); //Para todos los motores
        }
        if(p2 == 1){

            moveTorret(p3, p4); //Movimiento de la Torreta

        }else{

            moveCannon(p3, p4); //Movimiento del cañón
        }
    }else{

        tracksControl(p2, p3, p4); //Movimiento de las orugas
    }
}

```

Torreta:

En el método para mover la torreta hacemos el uso de la librería [AccelStepper](#), que nos permite girar nuestro motor de paso a paso de una manera sencilla.

Instanciamos un objeto de la clase [AccelStepper](#) a cuyo constructor le pasamos el modo (medio paso, más lento, pero más torque, y paso completo, más rápido, pero menos torque), y los pines de Arduino a los que lo hemos conectado.

Después, hay que especificar la [velocidad máxima y la aceleración](#). Una vez configurado, hacemos uso de una variable posición, que guarda la posición en la que se encuentra nuestro motor para sumarle, y que el movimiento sea contrario a las agujas del reloj o restarlo, para que sea en el sentido de las agujas del reloj. Le indicamos que esa es a la posición que se va a mover y con el método “[run](#)” se ejecuta el movimiento,

```
AccelStepper stepper(16, 8, 9, 10, 11);  
  
stepper.setMaxSpeed(1700);  
stepper.setSpeed(1700);  
stepper.setAcceleration(2000);  
  
position = position - 1;  
stepper.moveTo(position);  
stepper.run();
```

Cañón:

El movimiento del cañón hace uso de la librería [Servo.h](#) cuya función es facilitar el movimiento de un servomotor, el código usado es bastante similar al código del motor de paso a paso, con una variable que controla la posición en la que se encuentra.

La configuración es **mucho más sencilla** puesto que solo hay que instanciar el objeto servo e indicarle el pin de datos al que está conectado.

```
Servo cannon;  
cannon.attach(13);  
  
if(posCannon < 96.0){  
    posCannon = posCannon + 0.1;  
    aux = round(posCannon);  
    cannon.write(posCannon);  
}
```

Orugas:

Para el control de las orugas no hay que instalar **ninguna librería** puesto que solo tenemos que escribir algunos pines, en modo analógico, indicándoles la potencia. Por ejemplo, para ir hacia delante tenemos que **escribir los pines que están asociados al movimiento hacia delante**, en este caso el pin 4 y el pin 3. Este es el esquema de conexiones:

Pin 2: motor izquierdo hacia detrás.

Pin 3: motor izquierdo hacia delante.

Pin 4: motor derecho hacia delante.

Pin 5: motor derecho hacia detrás.

```
if(p4 == 0){  
  
    analogWrite(5, 0);  
    analogWrite(4, 25);  
  
    analogWrite(3, 25);  
    analogWrite(2, 0);  
}
```

6. Conclusiones:

Una vez terminado el proyecto hemos llegado a la conclusión de que nuestros estudios no solo sirven para hacer una aplicación de móvil o de ordenador, sino que se puede hacer cualquier cosa que te puedas imaginar.

Gracias a este proyecto hemos aprendido a que podemos integrar cualquier tipo de tecnología por muy diferentes que sean, por ejemplo, nosotros hemos conseguido integrar Arduino con un juego de Unity, e incluso se podría integrar con una aplicación de escritorio o de Android.

7. Dificultades

La mayor dificultad a la que nos hemos enfrentado ha sido la creación de la maqueta en general, ya que, a la hora de comprar componentes electrónicos, en algunos casos ha tardado un mes el envío, por lo que retrasaba el avance de algunas partes del proyecto.

Otro de los problemas a los que nos tuvimos que enfrentar fue la sincronización entre la velocidad de movimiento de la maqueta con la velocidad de movimiento en Unity, fue un trabajo tedioso de prueba y error que podría haberse arreglado con un acelerómetro y un giroscopio.

8. Bibliografía

Motor Unity

Unity download, requisitos de sistema

Vuforia

MC-1 USSR Toon Tank

Unity Particle Pack 5.x

Example_Gun

Military target

Simple Modern Crosshairs:

Spitfire.

ServoMotor.

Stepper:

Codecs de video

Uln2003:

<http://inteligenciaartificialyrobotica.com/esp/item/472/driver-uln2003-modulo>

<https://www.iberobotics.com/producto/driver-control-motor-paso-a-paso-uln2003-stepper/>

Arduino:

<https://www.xataka.com/makers/empezar-arduino-que-placa-kits-iniciacion-comprar#:~:text=Las%20diferencias%20entre%20una%20placa,por%20ejempl o%20la%20memoria%20interna.>

<https://arduino.cl/que-es-arduino/>

<https://www.fundacionaquae.org/sabes-arduino-sirve/>

<https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno>

<https://www.arduino.cc/reference/es/>

Motor de paso a paso:

<https://www.instructables.com/Motor-Driver-BTS7960-43A/>

<https://www.luisllamas.es/controla-motores-de-gran-potencia-con-arduino-y-bts7960/>

Baterías:

<https://www.dronetrest.com/t/everything-you-need-to-know-about-lipo-battery-chargers/1326>

https://es.wikipedia.org/wiki/Bater%C3%ADa_de_pol%C3%ADmero_de_litio

HUD:

<https://docs.unity3d.com/es/2019.4/Manual/UICanvas.html>

<https://docs.unity3d.com/es/2018.4/Manual/class-Canvas.html>.

<https://docs.unity3d.com/es/2018.4/Manual/script-CanvasScaler.html>

Diana:

<https://docs.unity.cn/es/2018.4/Manual/class-MeshFilter.html>

<https://docs.unity.cn/es/2018.4/Manual/class-MeshRenderer.html>

Raycast:

<https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>

<https://docs.unity3d.com/ScriptReference/RaycastHit.html>

<https://docs.unity3d.com/2020.1/Documentation/Manual/class-ArticulationBody.html>

<https://academiaandroid.com/colliders-deteccion-colisiones-juego-unity-3d/>

<https://docs.unity3d.com/ScriptReference/Rigidbody.html>

GameObject:

<https://docs.unity3d.com/es/2018.4/Manual/class-GameObject.html>

<https://gamedevtraum.com/es/desarrollo-de-videojuegos/tutoriales-y-soluciones-unity/serie-fundamental-unity/unity-que-es-un-gameobject/>

<https://www.digitallearning.es/intro-programacion-js/objetos-propiedades-metodos.html>

<https://riptutorial.com/unity3d/example/5798/intro-to-quaternion-vs-euler>

https://es.wikipedia.org/wiki/Bloqueo_del_card%C3%A1n

<https://docs.unity3d.com/ScriptReference/Object.Instantiate.html>

Transform:

<https://docs.unity3d.com/ScriptReference/Transform.html>

<https://docs.unity3d.com/ScriptReference/Transform.Rotate.html>

<https://docs.unity3d.com/ScriptReference/Transform.LookAt.html>

Motores:

<https://www.cursosaula21.com/como-funciona-un-motor-electrico/>

https://www.banggood.com/Machifit-775-795-895-MotorMotor-Bracket-DC-12V-24V-3000-12000RPM-Motor-Large-Torque-Gear-Motor-p-1342261.html?rmmds=myorder&cur_warehouse=CN&ID=511154