Universal Ranged Weapon System Documentation

Eepy Interactive

Contents

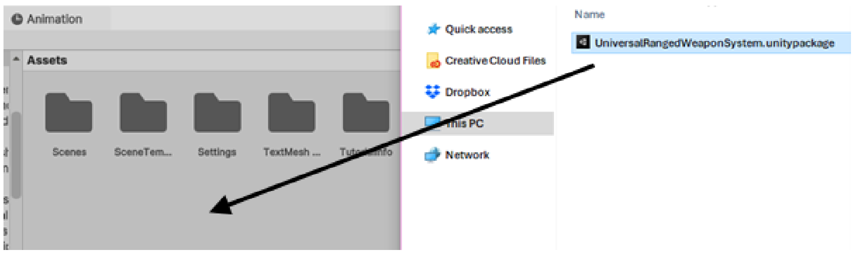
Disclaimer

The following documentation serves as a useful tool in your development with this package. It is by no means required to read the whole thing, but may help you better understand this package and its intentions. There is a setup guide and explanation for what each component adds to your game's weapons.

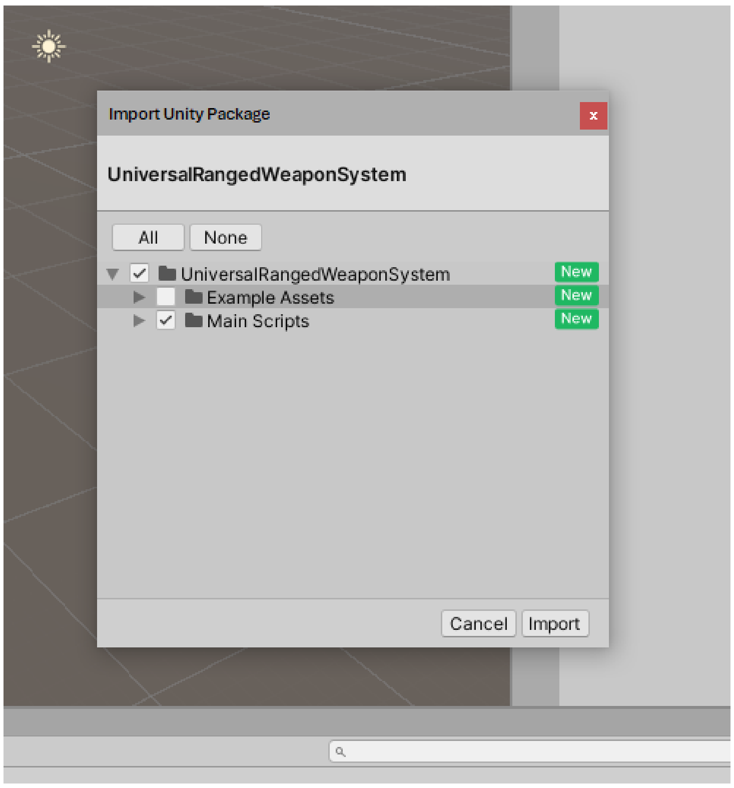
Setup

Importing the Package

Install from the Asset Store and package manager or drag and drop the .unitypackage file into your project folder.



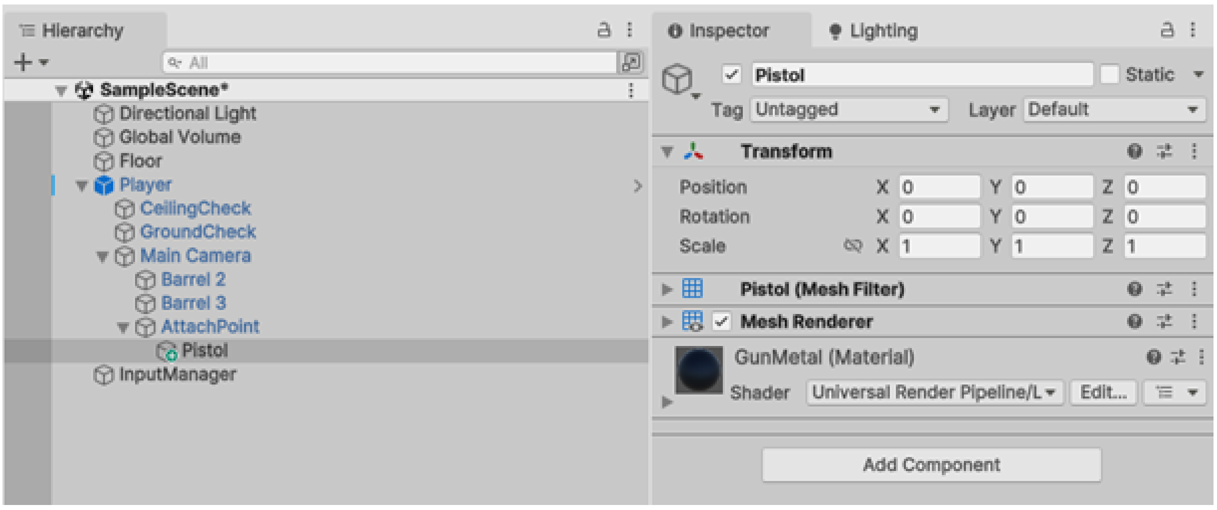
You will see the option to include "Example Assets" and "Main Scripts". **Main Scripts are required** for the package to work, but you may wish to exclude examples for your project. If you have not used this package before, it is **recommended to include the examples** – you can always delete them later.



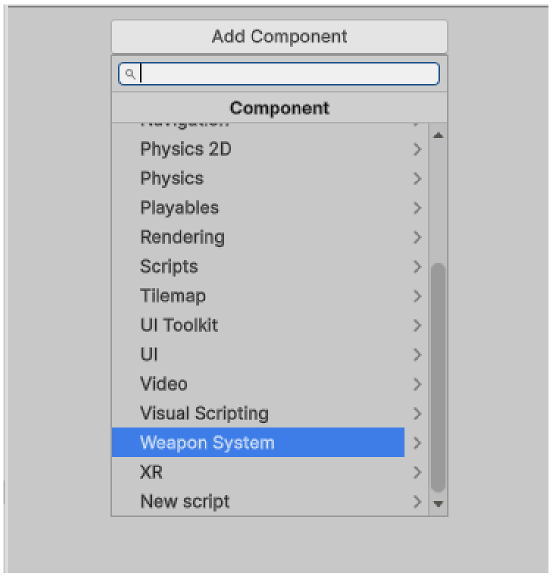
See Here for full details about example scripts and issues importing them.

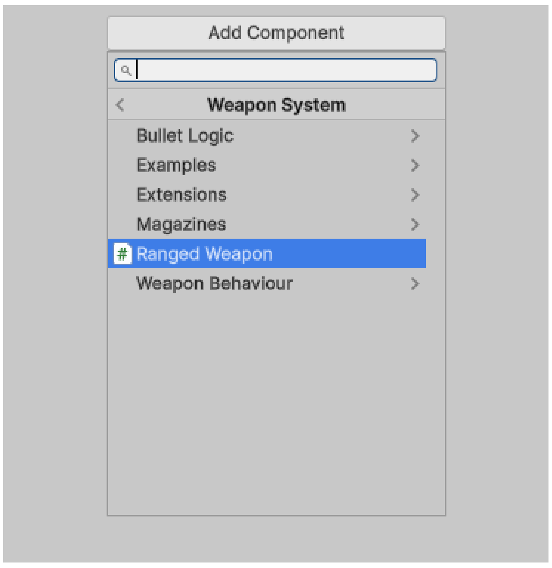
Attaching components

Navigate to your weapon model



Attach the "Ranged Weapon" component





This will add some required modules for you.

Weapon Controllers

If you are using the example scripts, now is a good time to also attach the **Basic Player Weapon Controller** to the gameObject. Note that this requires the **Input Manager** prefab to be in the scene.

If you are not using examples, then you will need to create a controller that handles the following functions:

- OnTriggerDown()
- OnTriggerUp()
- TryReload()

Now that the component is attached to the gameObject, the weapon parameters can be adjusted to the desired functionality.

Configuring Required Modules

Note:
Since you can’t remove required modules, you cannot simply remove the default modules. Instead, add the replacement module to the gameObject first; the old component will remove itself.

Bullet Logic

This controls the approach and behaviour of the "bullet" itself. Mostly to distinguish between a physics based projectile, or a mathematical ray/hitscan approach.

- **Bullet Origin Transform**

  Where the bullet is spawned. It is recommended to use the camera/head for hitscan and the nozzle of the weapon for projectiles.

Hitscan Parameters

Hitscan draws a line directly forward until it hits an object. This calculation is done in one frame, allowing for instantaneous bullets without collision issues.

- **Hitscan Layer Mask**
  Select the layers that the raycast should consider as a ’hit’. All other layers are ignored; the raycast will pass through them.

- **Decal Prefab**
  The object that will be spawned where the raycast hits. If there are multiple raycasts, multiple prefabs will be spawned.

- **Max Concurrent Hit Decals**
  The maximum number of hitscan decals created by this specific weapon in the scene before decals begin getting reused by the Pooler. Zero indicates there is no maximum, and the Pooler will not be used.

Penetrating Hitscan

Penetration refers to the raycast hitting an object and continuing past it, such as being able to shoot through a weak wall or through multiple enemies.

This inherits from the Hitscan component and therefore shares all its listed parameters. The following are unique to the Penetrating Hitscan component.

- **Penetration Layer Mask**
  Layers that the raycast will consider penetrable, ignores any values that are not on the hitscan layermask.

- **Penetration Max Count**
  The number of objects a bullet can pass through before stopping. Zero indicates that there is no limit, and it can pass through these layers indefinitely.

Projectile

Spawns a new object and fires it forward. Allows for gravity to be applied and allows the player to see the bullet moving. Useful for slower weapons that are not guns, such as bows.

- **Projectile Prefab**
  The projectile gameObject prefab which will be instantiated on weapon fire.
- **Projectile Force Direction**
  The axis on which the projectile will be fired relative to the forward of ’Bullet Origin Transform’.
- **Projectile Force**
  The magnitude of the force applied to the projectile when the weapon is fired.
- **Max Concurrent Projectiles**
  The maximum number of projectiles created by this specific weapon in the scene before projectiles begin getting reused by the Pooler. Zero indicates there is no maximum, and the Pooler will not be used.

Weapon Behaviour

This controls the way in which the weapon handles firing, such as how often the weapon fires and whether the trigger can be held down.

- **Minimum Time Between Shots**
  The time (in seconds) before the weapon can be reloaded or fired again.

Single Fire

For Single Fire weapons, the trigger needs to be individually pressed for each fire.

Burst Fire

Burst Fire is similar to Single Fire but, the weapon fires multiple times for each trigger.

- **Time Between Burst Shots**
  The time (in seconds) between fires DURING the burst.
- **Fire Count**
  The number of times the weapon is sequentially fired in a burst.
- **Consume Full Ammo**
  Whether a fire should consume the full proper amount of ammo. When enabled, each fire of the weapon will consume one bullet, if disabled then only one bullet is used in a burst.

Auto Fire

The weapon continuously fires in intervals when the trigger is held down.

Magazine

Magazines describe the way in which ammo is stored in the weapon.

- **Max Ammo Count**
  The maximum number of bullets that can be held. Zero indicates that there is no limit.
- **Start With Ammo**
  Whether the weapon should start with full ammo, else weapon will have no ammo when starting.

Basic Magazine

Basic Magazine holds ammo in groups (a magazine), once a group has depleted the weapon must be reloaded before it can continue firing.

- **Magazine Ammo Capacity**
  The number of bullets in a magazine such that a magazine size of 1 requires a reload after every fire
- **Is Ammo Persistent?**
  Should the weapon remember the bullets in the current magazine when reloading. Turning this off results in bullets being wasted/lost.
- **Reload Time Duration**
  The time (in seconds) that it takes for the reload to take effect and the weapon to be functional again.

Bullet Well

This preset essentially ignores magazines. It will hold all the bullets in a single magazine remove the need to ever reload.

Optional Extensions

Ejection

Ejection refers to firing out an empty shell when the weapon is fired, this is purely visual. It is included in the system so that spawned objects can be pooled and properly managed.

- **Empty Shell Prefab**
  The prefab that will be instanced at Shell Exit Point.
- **Shell Exit Point**
  New prefabs will be spawned here with this object’s world rotation.
- **Ejection Force**
  The magnitude of the impulse force applied to shell.
- **Max Concurrent Ejected Shells**
  The maximum number of ejected shells created by this specific weapon in the scene before shells begin getting reused by the Pooler. Zero indicates there is no maximum, and the Pooler will not be used.

Multi-Barrel

Fires from multiple origin points, as if the weapon has multiple barrels. When enabled, multiple raycasts will be drawn from each selected point.

- **Additional Barrels**
  These act as additional "Bullet Origin Transform" for any extra barrels added.
- **Cycle Between Barrels**
  When enabled the barrels will be fired in order (starting with Bullet Origin Transform). When turned off they will all fire at once, every single fire.

- **Consume Full Ammo**
  Whether firing the weapon should consume the full proper amount of ammo. When enabled, each barrel of the weapon will consume one bullet, if disabled then only one bullet is used per fire.

Note: Consuming Full Ammo is only relevant when "Cycle Between Barrels" is false.

Overheating

Overheating refers to weapons "jamming" or being otherwise unable to fire if used too often. When the limit is reached, the weapon will disable itself for a brief period as if overheating.

- **Heat Per Fire**
  The amount of "heat" added to the weapon every time it fires. Its overheat limit is arbitrarily set to 10.

- **Cooldown Delay**
  The time (in seconds) to begin the cooldown.

- **Cooldown Time**
  Cooldown refers to the weapon's heat returning to normal after not firing. The time (in seconds) for the weapon's heat to return to 0. This value will be used relative to how much heat it currently has.

- **Disable Time**
  The time (in seconds) that the weapon should be disabled for once the weapon has overheated.

Recoil

Some recoil settings inherit from the selected Recoil Settings Asset.

- **Recoil Cooldown Speed**
  The speed (degrees per second) at which the recoil effect is gradually undone and returns to normal.

- **Recoil Type**
  Whether or not the calculated recoil should affect the trajectory of the bullets. Calculated recoil can be accessed from the weapon regardless. Adding inaccuracy will defect the fired bullet in the direction of the inaccuracy.

Random Direction

Random Direction will increase recoil in a new random direction each fire. The greater the recoil the further from the centre the direction might be.

- **Recoil Directions**
  Which Direction(s) should the recoil be allowed to deviate in.

- **Axis Bias**
  An axis separated multiplier for the recoil (values between 0-1 recommended). Setting values to 1 indicates regular recoil intensity, lower values mean less intensive recoil.

- **Per Fire Multiplier**
  Equivalent to the angle (in degrees) added to the recoil per shot.

- **Max Inaccuracy Angle**

The angle (in degrees) between the starting direction and the furthest possible vector from this point.

Vector Array

Allow the weapon to specify exactly how the recoil should act over time by plotting deviance values in an array. This mechanic is similar to how games like Counter Strike apply recoil.

- **Recoil Vector Path**
  Use this to plot out exactly how the recoil should behave, values may be interpolated.

- **Recoil Vector Path Loop Start Index** It is recommended to have a number of the last values loop so that there is no clear end to this finite list. This variable indicates the first index in this loop, and will be the next position used after the last position is reached.

Shotgun

This extension modifies the number of projectiles/raycasts that are sent out each fire of the weapon; akin to how a shotgun fires out a large number of pellets rather than a whole bullet.

- **Bullets Per Fire**
  The number of projectiles/raycasts that are sent out each fire of the weapon.

- **Max Spread Angle**
  The angle (in degrees) between the centre fire line and the edge of the cone of fire.

Warmup

A warmup is a delay between beginning to hold down the trigger and the weapon beginning to fire. Consider this as the weapon charging up.

- **Time Duration**
  The time (in seconds) for the weapon to finish the warmup sequence and begin firing.

- **Is Cancellable?**
  If the trigger is let go before the warmup has completed, should the weapon cancel the warmup?

Object Pooling

While completely optional, object pooling is a more efficient way to handle lots of objects that need to be spawned and killed. It recycles objects as needed rather than creating new ones every time. The only reason not to use this is if you do not wish to delete the objects in question.

Maximum Concurrent *Object* – This is the maximum number of objects (of this type, from this weapon) that should exist in the scene. For example, if the limit is set to 10 and there are already 10 of these objects in the scene, when a new one is requested the oldest object in the scene will be recycled and used as the newly spawned one.

Note: Limits are handled by type, and per weapon. Meaning if you have a weapon prefab with a maximum of 30 Projectiles and 20 Shells (ejection extension). That is 50 objects per gun, with 10 of these guns in your scene that is a maximum of 500 objects in your scene. Even when pooling, consider your performance requirements.

If you need a global limit, it might be worth implementing a Pooler of your own (or implementing the "Pool" class provided in a global fashion).

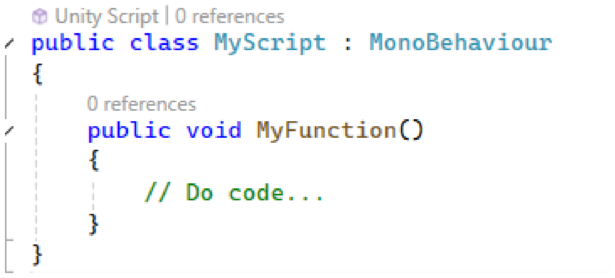
Events

The Universal Ranged Weapon System uses several Unity Events to broadcast important information to you the developer. With the provided events, it should be possible to create many additional features on top of this script without having to edit the original file.

There are two ways to utilise these events, through the inspector, and through code.

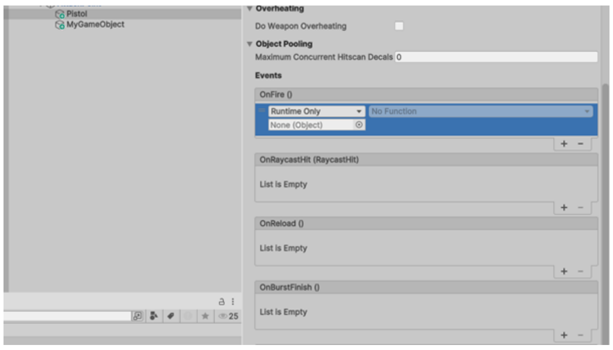
Implementing Via the Inspector

You will need to create a **public** function in a MonoBehaviour script. Then open the inspector.



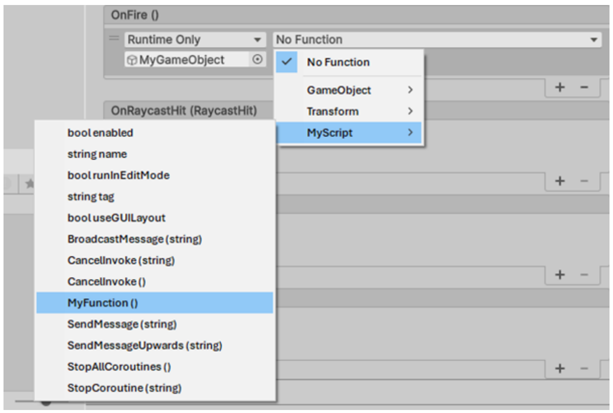
```
Unity Script | 0 references
public class MyScript : MonoBehaviour
{
    0 references
    public void MyFunction()
    {
        // Do code...
    }
}
```

Locate the event you wish to subscribe to, then create a new action by hitting the plus icon. Then drag the object that holds your script into the empty box.



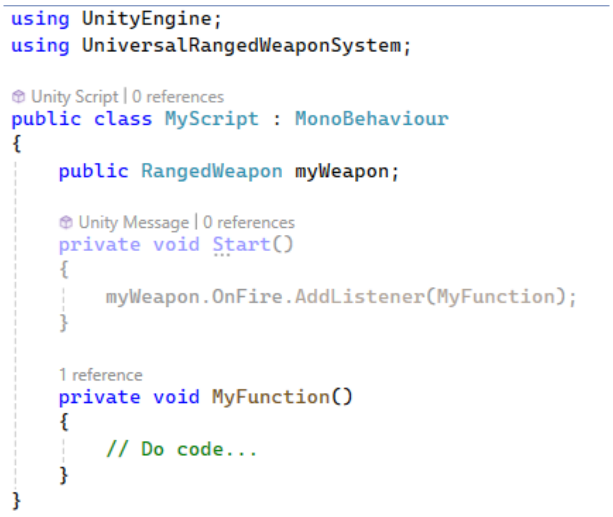
Overheating
Do Weapon Overheating
Object Pooling
Maximum Concurrent Hitscan Decals 0
Events
OnFire ()
Runtime Only | No Function
None (Object)
OnRaycastHit (RaycastHit)
List is Empty
OnReload ()
List is Empty
OnBurstFinish ()
List is Empty

Using the dropdown menu, select your script, and then the function on that script you wish to call.



Implementing Via Code

Unlike when implementing through the inspector, the function to call does not need to be public. Start by getting a reference to the weapon you are subscribing to. To do this, ensure you include the *UniversalRanged-WeaponSystem* namespace.



```
using UnityEngine;
using UniversalRangedWeaponSystem;

Unity Script | 0 references
public class MyScript : MonoBehaviour
{
    public RangedWeapon myWeapon;

    Unity Message | 0 references
    private void Start()
    {
        myWeapon.OnFire.AddListener(MyFunction);
    }

    1 reference
    private void MyFunction()
    {
        // Do code...
    }
}
```

It is good practice to name “MyFunction” after the event that it’s subscribed to, so in this example it should be called “OnFire”.

Example Scripts

To demonstrate how to best utilise this package, there are several optional example scripts that help show how this package might be used in a larger project. They can also be used to quickly set up a working prototype so that parameters can be easily iterated on.

Note:
The example scripts are based on a URP template project in Unity 6000.0.59f2. You may run into dependency issues when using Example Assets if you do not have the Universal Render Pipeline, or Text Mesh Pro installed in your project. (Text Mesh Pro now comes pre-installed as of Unity 2020.1)

Basic Player Weapon Controller

Passes input events onto the associated Ranged Weapon. Attach this to your player object to be able to control your created weapon.

This demonstrates how to utilise events in your own scripts, such as input or enemy AI logic and pass it onto the weapon script.

- **Ranged Weapon**
  The Ranged Weapon component that this controller should control.

Input Manager

This is a Singleton class that is used to manage player inputs from within any script. Several example scripts use this to detect input from the player. Example scripts depend on it being in the scene so it is recommended to add the prefab of the same name to your scene.

Player Camera Controller

Add the ability to move the camera around with the mouse. Adapted from the script provided by Unity in the FPS template.

Ranged Weapon exposes a public variable called Recoil Rotation. This shows how you should rotate the player camera in a way that is compatible with weapon recoil. It uses 'transform.Rotate()' rather than setting the rotation outright, allowing other scripts to also affect camera rotation.

Without this, the example scenes would not allow the player to look around and properly use the guns.

- **Mouse Sensitivity**
  How much to rotate the camera when the mouse moves. Higher values mean the camera rotates faster.

- **Player Body**
  The GameObject that is considered “the player”, usually the parent of the camera this component is attached to.

- **Recoil Smooth Speed**
  The speed at which the camera will move to the new position as set by the recoil.

Player Movement

Adds basic movement to a player, adapted from the script provided by Unity in the FPS template.

- **Speed**
  The speed of player movement in metres per second.

- **Gravity**
  The acceleration that is applied to the player when they are falling to the ground, in metres per second squared.

- **Jump Height**
  The force applied to the player when jumping.

- **Step Offset Override**
  Overrides the Step Offset of the attached Character controller so that it can set the Step Offset to zero when in the air.

- **Ceiling/Ground Check**
  This should be set to an empty object that is at the player's head/feet. Used to validate whether the player is touching the ceiling/ground while jumping.

- **Ground Distance**
  The distance the player can be from the ground before being considered in the air. (Helps with steps and slopes).

- **Ground Mask**
  A Layer Mask of all layers that should be considered the ground.

Projectile

Attach this component to a GameObject to make it act like a basic projectile that gets destroyed on impact. (Note that projectiles used with a Ranged Weapon should be prefabs and not objects within a scene).

An example projectile that, if needed, can be used to work with wider game systems.

- **Alive Time**
  The time in seconds that this object will exist in the scene for before disabling itself.

- **Collision Layer Mask**
  A Layer Mask of all layers that should destroy this object on collision. Leave this blank if you don't want that behaviour.

Weapon HUD

A controller for HUD elements related to the Weapon. This is meant to be a visual indicator of your weapon properties working rather than a finished game-ready HUD.

- **Ranged Weapon**
  The ranged weapon that the HUD should read from.

- **HUD UI images**
  UI images for the HUD, these will be scaled and manipulated as needed.

- **Ammo Text**
  The text component that displays relevant info about how much ammo is in the weapon.

  This will automatically change format to make the magazine type attached. For example basic magazine displays in the format

  > Ammo in Magazine | Magazines Left

  unless there is no limit to magazines - then it displays only the ammo in the current magazine.

Weapon Pad

A component used to assign a specific weapon to the player’s Weapon Socket when standing on it. This is useful in the demo scene to allow developers to test all the different weapon prefabs available.

Demonstrates how to use the Weapon Socket component.

- **Weapon Prefab**
  The prefab object that is to be assigned to the player while standing on the Weapon Pad.

Weapon Socket

A component that manages all the other example components that rely on a reference to a weapon. By attaching the weapon to this component, all the dependant components will be updated. This allows the weapon to be changed at runtime much easier. (Note that Null is a valid input for assignment and indicates that no weapon is being held).

This component shows how a Weapon can be used in wider game-systems that require changing the used weapon at runtime.

- **Bullet Origin Transform Override**
  Assign this if you want to override all weapons to a specific Bullet Origin such as the player camera.

  Can be left blank if the prefab has already assigned this parameter on the weapon, which is not always feasible.

Weapon Sounds

Adds SFX to the weapon fire. Depicts how to subscribe to events via code.

- **Sound Effect**
  The sound clip to be played every time the gun fires.