# Weapons

This page describes the mechanics behind the TopDown Engine's weapons, and how to create and use your own weapons.

## Introduction

The TopDown Engine includes a **generic weapon system**, allowing your characters to equip a weapon (projectile based, melee, or whatever you can think of), switch to another one, and use it. The engine also includes a few examples of weapons you can have a look at, and that you can use as a basis when creating your own.

## The CharacterHandleWeapon Ability

To be able to equip and use a weapon, a Character must have a **CharacterHandleWeapon** component. From its inspector you'll be able to optionally select a weapon for your character to start with, define if it can pickup new ones, and specify a **Weapon Attachment**. This is a transform on your character (or nested inside its prefab) for the weapon to attach to. If you don't specify one, the weapon will attach to the root level of your prefab.

What the **CharacterHandleWeapon** component will then do is check for changes on the weapon use button, and if it gets pressed/released, transfer that information to the weapon currently equipped. From its inspector you can also define input **buffering** parameters, which will impact how long input requests last (the longer they do, the longer the system will "remember" that you want another weapon use to happen).

## The Weapon classes

All weapons in the TopDown Engine derive from the **Weapon** class. You can of course divert from that, but that's how all the examples work. The Weapon class is meant to be extended and will define a lot of things common to all or most weapons. Using it as a basis you'll be able to create everything, from a shotgun to a rocket launcher, but also a katana or a grappling gun. In addition to providing a solid basis for animations, sounds, state management, it allows you to define in your child class what happens when the weapon is used, and build from there. You can look at the **ProjectileWeapon** and **MeleeWeapon** classes for examples of how you can create very different weapons from this very same script.

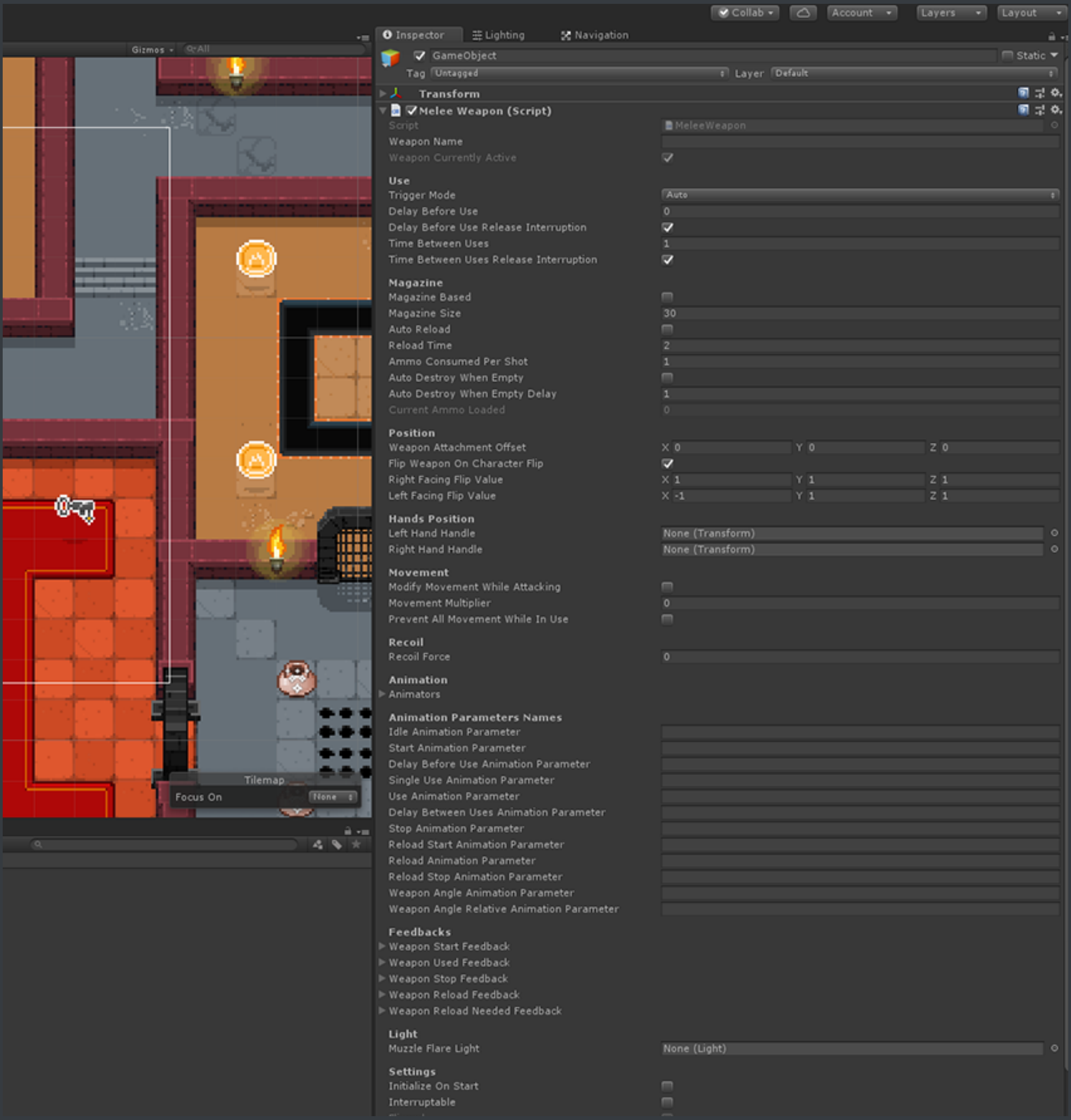
## Creating a weapon

To create your own weapon, you can either reuse the ProjectileWeapon or MeleeWeapon scripts as they are, extend them, or create your own Weapon child class. In any case, you'll then need to **create a weapon prefab**. To do so, you'll need a gameobject. Basically you'll have the choice to have a visible weapon or an invisible one.

Once you have your sprite, or model, or empty game object, just add your weapon (ProjectileWeapon, MeleeWeapon, or your own) script to it. From its inspector you'll be able to setup animations, sounds, effects to trigger when firing, and specify how the weapon should flip when the character flips.

Here's a step by step creation of a Melee Weapon for reference :

- create an empty gameobject
- add a MeleeWeapon component to it



GameObject — Static
Tag: Untagged — Layer: Default

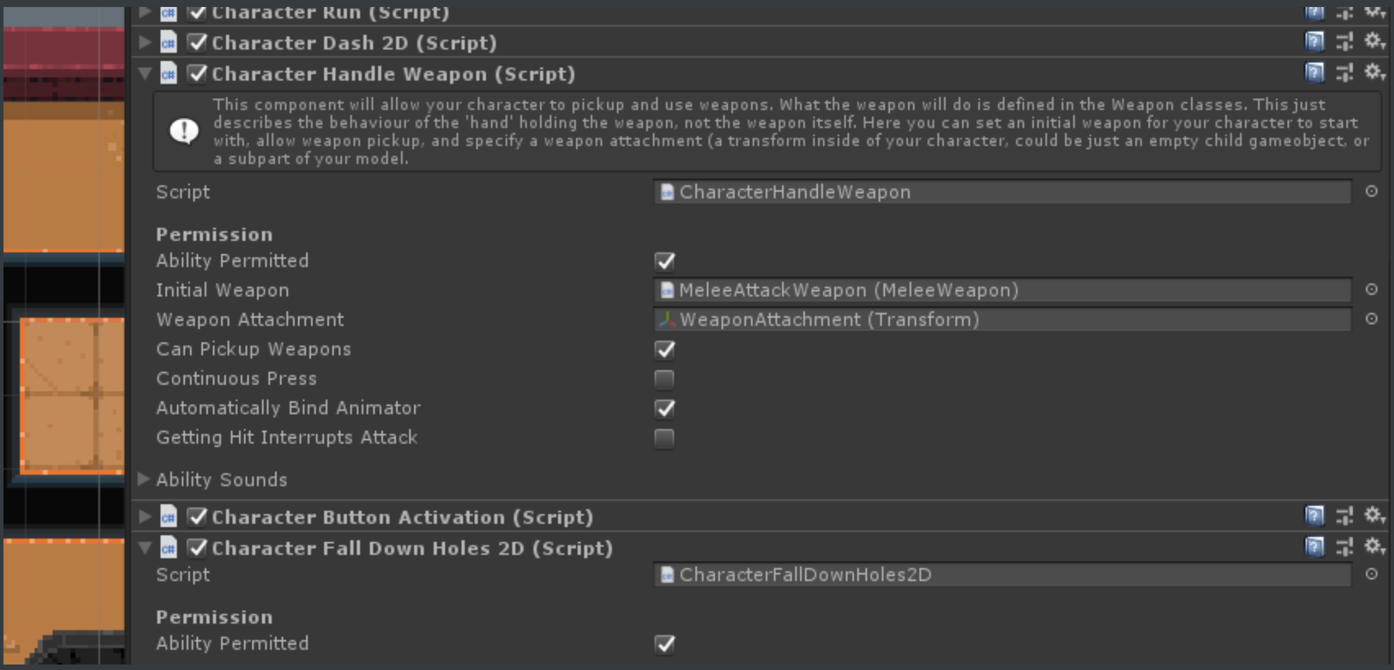
Transform

Melee Weapon (Script)

| | |
|---|---|
| Script | MeleeWeapon |
| Weapon Name | |
| Weapon Currently Active | ✓ |
| **Use** | |
| Trigger Mode | Auto |
| Delay Before Use | 0 |
| Delay Before Use Release Interruption | ✓ |
| Time Between Uses | 1 |
| Time Between Uses Release Interruption | ✓ |
| **Magazine** | |
| Magazine Based | ☐ |
| Magazine Size | 30 |
| Auto Reload | ☐ |
| Reload Time | 2 |
| Ammo Consumed Per Shot | 1 |
| Auto Destroy When Empty | ☐ |
| Auto Destroy When Empty Delay | 1 |
| Current Ammo Loaded | 0 |
| **Position** | |
| Weapon Attachment Offset | X 0 Y 0 Z 0 |
| Flip Weapon On Character Flip | ✓ |
| Right Facing Flip Value | X 1 Y 1 Z 1 |
| Left Facing Flip Value | X -1 Y 1 Z 1 |
| **Hands Position** | |
| Left Hand Handle | None (Transform) |
| Right Hand Handle | None (Transform) |
| **Movement** | |
| Modify Movement While Attacking | ☐ |
| Movement Multiplier | 0 |
| Prevent All Movement While In Use | ☐ |
| **Recoil** | |
| Recoil Force | 0 |
| **Animation** | |
| Animators | |
| **Animation Parameters Names** | |
| Idle Animation Parameter | |
| Start Animation Parameter | |
| Delay Before Use Animation Parameter | |
| Single Use Animation Parameter | |
| Use Animation Parameter | |
| Delay Between Uses Animation Parameter | |
| Stop Animation Parameter | |
| Reload Start Animation Parameter | |
| Reload Animation Parameter | |
| Reload Stop Animation Parameter | |
| Weapon Angle Animation Parameter | |
| Weapon Angle Relative Animation Parameter | |
| **Feedbacks** | |
| Weapon Start Feedback | |
| Weapon Used Feedback | |
| Weapon Stop Feedback | |
| Weapon Reload Feedback | |
| Weapon Reload Needed Feedback | |
| **Light** | |
| Muzzle Flare Light | None (Light) |
| **Settings** | |
| Initialize On Start | ☐ |
| Interruptable | ☐ |

Our empty game object with the MeleeWeapon component

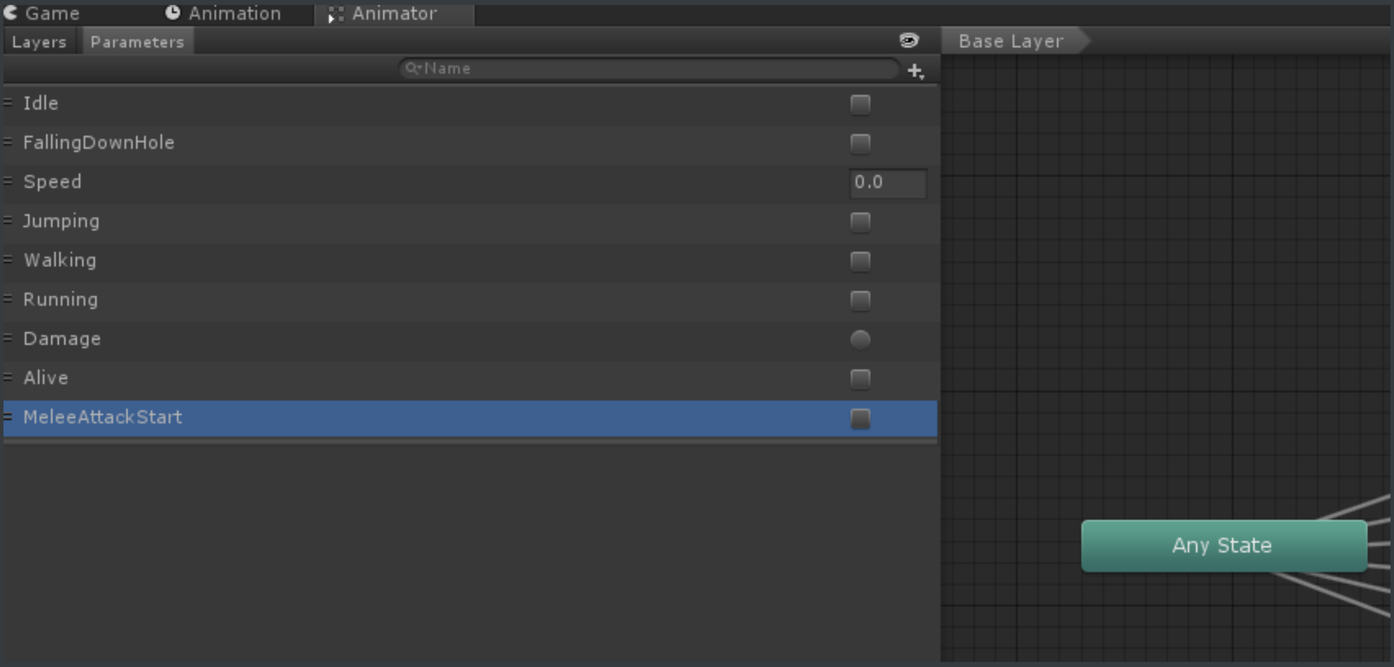
- in the MeleeWeapon inspector, set the following parameters : Trigger Mode to Semi auto, Damage Area Shape to Rectangle, Area Size to {1,1}, Area Offset to {2,0} (so that the damage area will activate in front of our character), Active Duration to 0.2 (it's a short attack, a punch basically). Then set the Damage Caused Target Layer mask to Enemies (and anything else you want to cause damage to), and Damage Caused to 10.
- rename this gameobject to MeleeAttackWeapon (or whatever you prefer)
- drag this gameobject into your hierarchy to make a prefab out of it
- select your character, and drag that newly created prefab into its InitialWeapon slot (or you can use the ChangeWeapon method via script to equip it)



```
Character Run (Script)
Character Dash 2D (Script)
Character Handle Weapon (Script)
This component will allow your character to pickup and use weapons. What the weapon will do is defined in the Weapon classes. This just describes the behaviour of the 'hand' holding the weapon, not the weapon itself. Here you can set an initial weapon for your character to start with, allow weapon pickup, and specify a weapon attachment (a transform inside of your character, could be just an empty child gameobject, or a subpart of your model.
Script                            CharacterHandleWeapon
Permission
Ability Permitted                 ✓
Initial Weapon                    MeleeAttackWeapon (MeleeWeapon)
Weapon Attachment                 WeaponAttachment (Transform)
Can Pickup Weapons                ✓
Continuous Press                  ☐
Automatically Bind Animator       ✓
Getting Hit Interrupts Attack     ☐
Ability Sounds
Character Button Activation (Script)
Character Fall Down Holes 2D (Script)
Script                            CharacterFallDownHoles2D
Permission
Ability Permitted                 ✓
```

Binding the weapon to our character

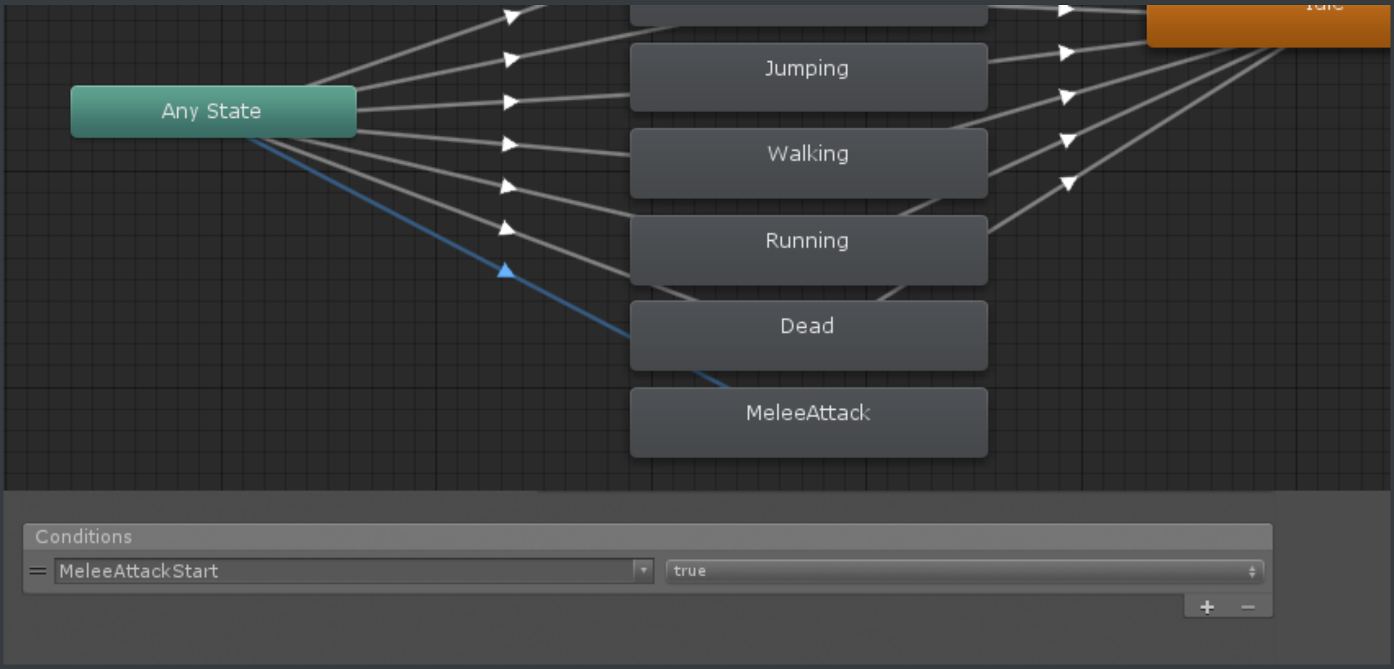
- that's it, you now have a working weapon. Press play, then E (by default), and you'll be throwing punches around and damaging enemies/objects. But so far you're probably not seeing anything when attacking. Let's bind an animation to that attack.
- select your character's animator, drag your attack animation into it if it's not already the case.
- create a new animation parameter (the little + button at the top right corner of the Parameters panel), in our example we'll call it "MeleeAttackStart"



Idle
FallingDownHole
Speed 0.0
Jumping
Walking
Running
Damage
Alive
MeleeAttackStart

Adding a new animation parameter

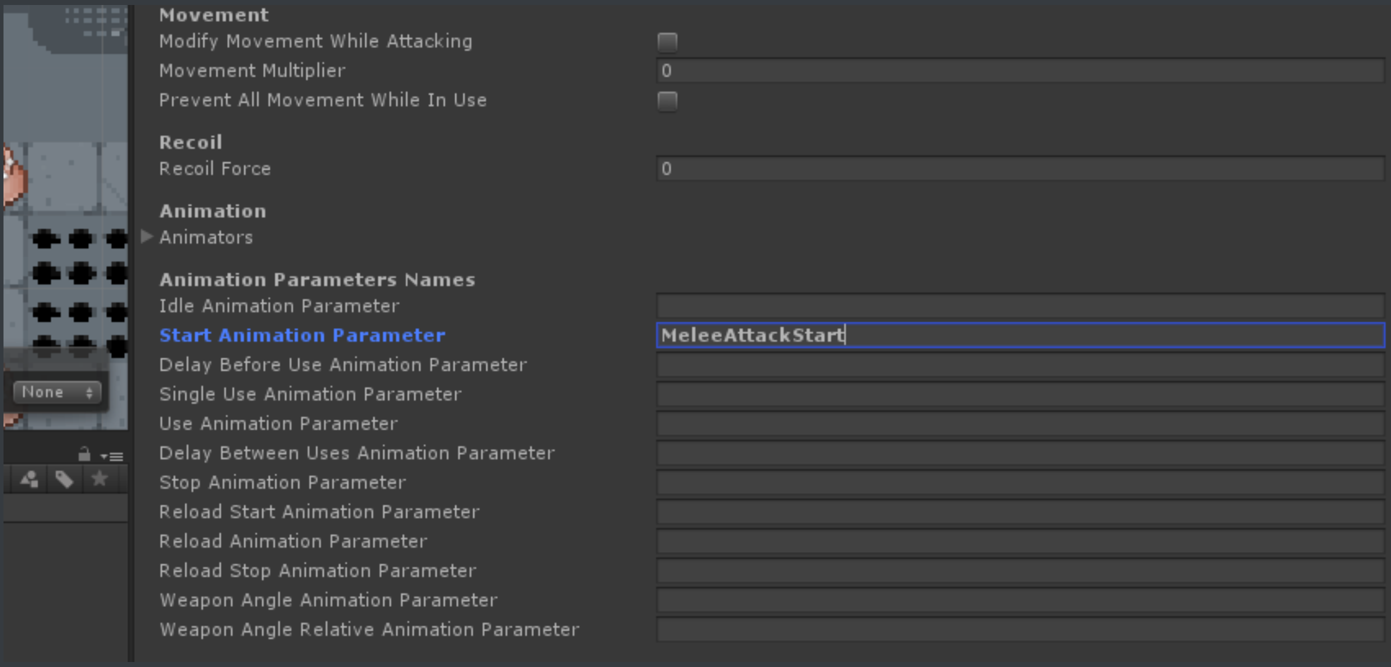
- create a transition to that animation.



Conditions
MeleeAttackStart true

Adding a new transition to our attack animation

- go back to your weapon's inspector, and fill the StartAnimation field with the name of your animation parameter. In our example "MeleeAttackStart".



Movement
Modify Movement While Attacking
Movement Multiplier: 0
Prevent All Movement While In Use
Recoil
Recoil Force: 0
Animation
Animators
Animation Parameters Names
Idle Animation Parameter
Start Animation Parameter: MeleeAttackStart
Delay Before Use Animation Parameter
Single Use Animation Parameter
Use Animation Parameter
Delay Between Uses Animation Parameter
Stop Animation Parameter
Reload Start Animation Parameter
Reload Animation Parameter
Reload Stop Animation Parameter
Weapon Angle Animation Parameter
Weapon Angle Relative Animation Parameter

Adding a new transition to our attack animation

- press play again, attack with your weapon, you now see your character attacking. You can add other animations, for when the attack is in progress, ending, etc.

## Projectile Weapons

Projectile weapons will require an **Object Pooler component** attached to them. You'll need to add either a **SimpleObjectPooler**, or a **MultipleObjectPooler** to your weapon, it won't shoot anything otherwise. Object poolers are a way to recycle projectiles as they're fired, that way you don't instantiate/destroy objects as you go, which is much better for your game's performance. Simple object poolers will allow you to shoot one type of projectile only (which is just fine for most weapons), but if you want to get fancy and have it shoot different type of stuff, go for a MultipleObjectPooler. In your object pooler's inspector, drag the object you want to shoot in the GameObjectToPool field, define the pool size (the amount of items the game will store and reuse, how much that is really depends on your game, the level's size, etc. Basically the idea is that you want your pool to never be used entirely, but not be too large either). Finally you can decide whether the pool can expand to accomodate for edge cases where all the pool's been used but not recycled yet.

Additionally, you can define from the inspector the spread and amount of projectiles per shot. The spread is defined by a maximum angle (on all 3 axis), and can be random or not. If not random, you can combine it with multiple projectiles per shot to create patterns.

## Projectiles

Projectile Weapons usually shoot projectiles (I know). The aptly named Projectile class will handle most common uses, from bullets to rockets. It'll allow you to create projectiles that move in one direction (affected or not by the direction of the weapon), at a certain speed and acceleration.

To most projectiles you'll want to add a DamageOnTouch component, so they hurt their target, and potentially a Health component if you want them to be destroyable before they hit.

## Timing

Timing is essential when creating weapons, as you'll want to sync your weapon's component to its animation perfectly. All weapons come with two variables you can tweak directly from the inspector : **DelayBeforeUse** (the time between the input registration and the actual WeaponUse - the shot, axe sling, etc), and **TimeBetweenUses** (the time that needs to pass before you can have another WeaponUse (whether it's in auto, or via repeated presses, etc). The weapon just won't "shoot" during that time).

Additionally, the melee weapons have two more variables you can play with : **InitialDelay** (the delay between the WeaponUse and the activation of the damage area) and **ActiveDuration** (the time during which the damage area will stay active). These 2 last parameters are very specific to damage areas, and of course you'll want to sync these values with your DelayBeforeUse / TimeBetweenUses to prevent conflicts.

You can also set your weapons to prevent movement while attacking, making sure your character won't run around in the middle of its attack.

## Combo Weapons

The engine comes with built-in support for **combo weapons**, and they're super easy to create. You can look at the KoalaSword prefab for an example of how they work.

When using it repeatedly, a Weapon with a ComboWeapon component will **cycle** through all the Weapons added to that same object, in succession, and from top to bottom in the Inspector. If you've checked the **Droppable Combo** option, it'll be possible to "drop" the combo, meaning that if you fail to use your weapon again within the Drop Combo Delay the next weapon you use will be the first in line.

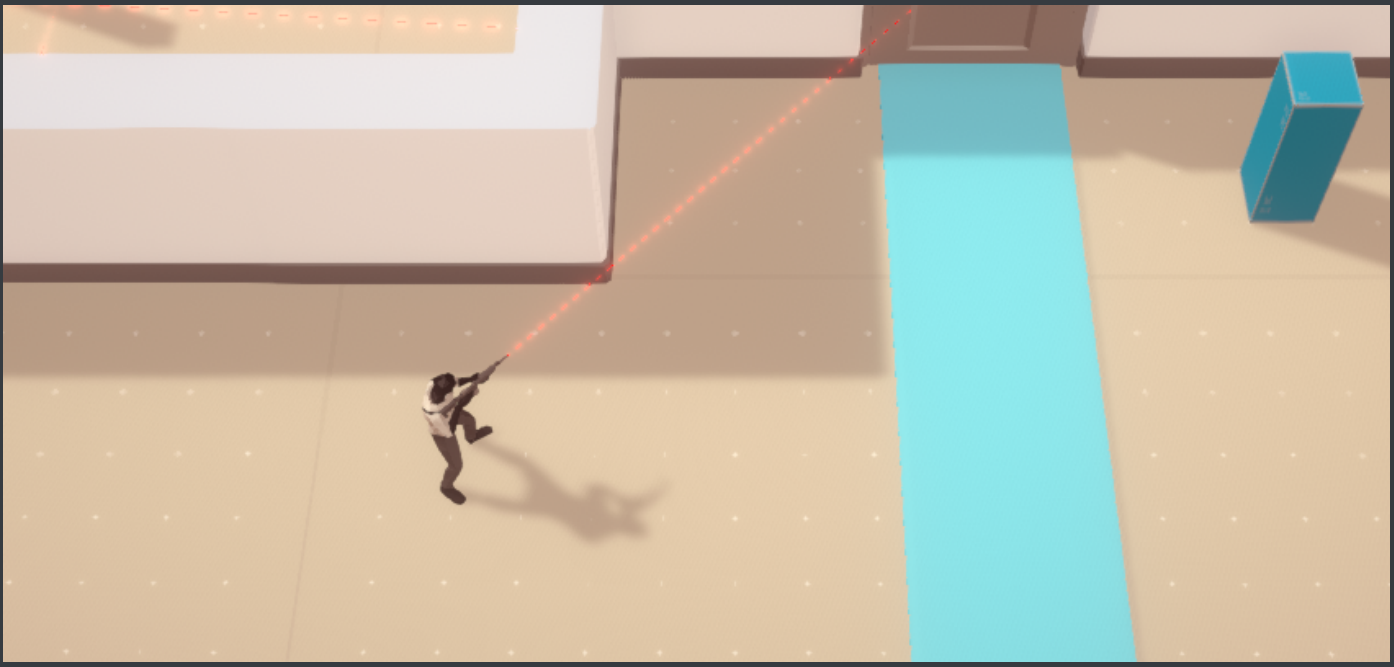
To create one, simply add multiple Weapon components (MeleeWeapons, ProjectileWeapons, or your own extensions of Weapons) to an object, and set each of them up like you would any other weapon. Then add a **ComboWeapon** component. In it, you just have to define whether or not the combo can be dropped, and after what delay (in seconds). A droppable combo means that if you wait too long (more than the specified delay) between two attacks, the next attack will be the first attack again, otherwise it'll move to the next in line.

## Secondary Weapons

Your character can equip more than one weapon at once, if desired. To do so, you can add a **CharacterHandleSecondaryWeapon** ability to it. It works exactly like CharacterHandleWeapon (it extends it), but looks at a different input. If needed, you could create more abilities like that (CharacterHandleThirdWeapon, etc), simply overriding the HandleInput method, following the CharacterHandleSecondaryWeapon example.

It also works with inventories. On your InventoryWeapon, you can optionnally define a **HandleWeaponID**. This is an int that corresponds to the ID specified in your target ability class. CharacterHandleWeapon is 1, CharacterHandleSecondaryWeapon is 2, and if you were to create more of these, you'd need to override and increment that int as well. A weapon with a HandleWeaponID of 2 will automatically go to your CharacterHandleSecondaryWeapon.

## Weapon Laser Sight



The lasersight component in action

You can add a **WeaponLaserSight** to your weapon. It's of course more suited for a projectile weapon but there are **no restrictions**. What it'll do is that it'll cast a laser in front of the weapon, like a laser sight would. You can set a collision mask so that the laser stops when meeting a wall or platform. From the inspector you can also customize the appearance of the laser.

## Weapon Aim

Adding a **WeaponAim** component (WeaponAim2D or WeaponAim3D) to your weapon will allow you to control its orientation, allowing your character to shoot up, down, etc. From its inspector you'll be able to define its control mode :

- **Off** : no weapon aim
- **PrimaryMovement** : the weapon will aim in whatever direction you're moving your movement stick/keys to.
- **SecondaryMovement** : the weapon will aim in the direction you point your secondary movement input to. That's the right stick on an xbox pad by default, for example.
- **PrimaryThenSecondaryMovement** : primary movement will be used if there's any, otherwise secondary movement
- **SecondaryThenPrimaryMovement** : secondary movement will be used if there's any, otherwise primary movement

- **Mouse** : the weapon will point in the direction of the mouse pointer
- **Script** : the weapon won't aim via regular input, but will be controlled by another script (useful for AIs, auto aim, etc)

This can be defined per weapon, but can also be forced at a global level, for all weapons, on the **InputManager**. Just set its WeaponForcedMode and it'll automatically be applied to all the weapons the player Character uses.

You can also define the **Rotation Mode** :

- **Free** : full 360° rotation
- **Strict 4 directions** : up, down, left, right
- **Strict 8 directions** : up, down, left, right, and the diagonals (up left, down right, etc)

Finally you can define a maximum and minimum angle to constrain movement to allow only up shots for example, or prevent the character to shoot in its back.

The WeaponAim component also allows you to display a **Reticle** on screen for your weapon. You'll need to specify a display mode (UI or scene), a gameobject prefab for it (there are two demo ones included in the asset, called SceneReticle and UIReticle, but you can use your own of course). You can set the distance from the character the reticle should be at, or if it should follow the mouse, if it should rotate as you aim your weapon, a dead zone (a distance from the character within which the reticle won't affect the aim), and whether or not the mouse pointer should be hidden. There are also options to have a scene UI reticle move along with slopes. From the inspector you'll also be able to define how the CameraTarget should move if you have a reticle active.

## Auto Aim

If you'd like your character to auto aim at enemies, you can add to your weapon a **WeaponAutoAim2D** or **WeaponAutoAim3D** component. This will require a WeaponAim2D/3D component on your weapon, set in Script mode (see above). In its inspector you'll be able to define a layermask containing the layers on which to look for targets, as well as one to consider as obstacle. You'll be able to set a radius and frequency at which to scan, rotation settings, how to impact the camera target, set an AimMarker prefab to highlight targets, and define feedbacks to play when a target is found, changes or when no more targets are found. You'll find examples of all that setup in the Colonel demo scene.

## Ammo

Thanks to the InventoryEngine, the engine natively supports **ammo** and the possibility to create **ammo based weapons**. There are two types of ammo based weapons you can create : inventory based, and regular ones.

**Inventory based weapons** will require your character to have an Inventory attached to it (via the CharacterInventory ability), while regular ammo based weapons will just consume infinite ammo, but both will allow you to define a magazine size, reload or not, etc.

All these settings can be set on the weapon's inspector. There you'll be able to define the magazine size, whether the weapon auto reloads when the magazine is empty, or if the player needs to press the reload key, how long the reload should take (in seconds), as well as how much of that ammo each shot consumes.

Checking the "Magazine based" checkbox will tell the weapon that it should take its ammo from the character's inventory. To do so, you'll also need another component on your weapon, WeaponAmmo. There you'll be able to specify the name of the ammo the system should look for in the inventory ("Bullets" for example), what the inventory's name is (usually MainInventory but you can pick any name you prefer as long as it's set that way on your inventory), the maximum ammo the weapon's ammo indicator should go for, and whether or not the weapon should load itself with ammo from the inventory when equipped.

## Weapons and Inventory

If your character is using an inventory, the Player can now **switch weapons** (cycling through all the weapons available in the inventory) using an input shortcut. By default this is bound to LB on an xbox pad, or t on a keyboard.

Additionally, if your character has a Weapon Inventory, and if that inventory contains a weapon when starting a level, this weapon will be automatically equipped.

## Weapons IK

From the Weapon's inspector you can define left and right hand attachments to use with Unity's built-in animation **IK system**. This will attach the hands of your character to specific points of your weapon, detach them when you unequip the weapon, and allow your character to move and aim naturally as you move your weapon. The demo characters in the Loft demo are all setup like that if you want a ready to use example.

Aside from creating your avatar properly (Unity's got great documentation about that), the only thing you'll need to ease up the process is the WeaponIK class, ideally set at the same level as your avatar's animator. This class allows for a 3D character to grab its current weapon's handles, and look wherever it's aiming. There's a bit of setup involved. You need to have a CharacterHandleWeapon component on your character, it needs an animator with IKPass active (this is set in the Layers tab of the animator) and the animator's avatar MUST be set as humanoid.

And for cases where you'd like to disable IK temporarily (during a dash roll, for example), you can add a **WeaponIKDisabler** on your animator, and you'll be able to specify a list of boolean animation parameter names that, when true, should cause IK to be turned off.

## Enemy Weapons

You'll find many examples of weapons used by enemies throughout the engine's demos. Creating one follows the **exact same process** as creating a Weapon for a Player Character. You usually can't simply copy a player weapon and put it on an enemy though. Depending on the weapon you'll likely want to change target layers, aim methods, etc, but the overall logic remains the same.

## Ammo Display

**AmmoDisplay** is a component you can add to a GUI rect to have it display a progress bar and text showing how much ammo you've got left in your magazine and inventory. Just add it to an empty GUI object, and in its inspector you'll need to bind it to a **ForegroundBar** and a Text object, and specify for what player you want to display that info (usually Player1). Finally, select your **GUIManager**, and drag your AmmoDisplay into its **AmmoDisplays** array.

## Weapon Models

While in many cases, you'll want to have your weapon's logic and visuals on the same object, sometimes you may want to separate both. The engine offers two different ways of doing that for inventory based weapons, where you want to be able to activate different models when equipping different weapons :

- The **WeaponModel** class is a class you can put on objects under your Character. From its inspector, you'll be able to set the name of a weapon that should activate this model. This name has to match the exact Weapon's item **ID**. Then you can have that model aim at the actual Weapon's aim direction, have it animate, override WeaponUse, rebind IK, and it even comes with MMFeedbacks hooks in addition to the ones already on the Weapon. Here you'll find simple steps to setup a WeaponModel, and you can find an example of it in action in the **Colonel** demo scene. That's how the Colonel's rifle works, look for the FBIAssaultRifleWeaponModel node on the Colonel prefab for reference.
- If you don't need that many features, you can also simply put a **WeaponModelEnabler** on your Character (at the top level, along with abilities, Health, etc). It's much simpler, and simply requires you define a list of nodes on your character that you want to enable when a certain weapon is equipped. To specify which weapon corresponds to what model, you need to put a **WeaponAnimationID**, which has to match the WeaponAnimationID you set on your weapon (under its Animation Parameters panel). You'll find steps to setup a WeaponModelEnabler in the Recipes section of the documentation, and an example of it in action on the LoftTie prefab, in the MinimalSword3D demo scene.

These two systems will let you enable (sub) models based on the current weapon, and should cover most use cases. Of course, you'll want to pick **one or the other**, not both.