Creator Kit: FPS

Additional technical documentation

This document is a reference for some parts of the Creator Kit: FPS. For a walkthrough on how to use the kit, see the tutorial documentation.

Manual

Creating levels and adding rooms

Level editing for the Creator Kit is controlled by a custom script called LevelLayout. You can connect or 'snap' together pre-created rooms to build levels.

The script relies on each room Prefab having a Level Room component enabled. This lists all the possible exits. These exits should be children objects with their z axis pointing out of the room. The exits (or doors) should all connect together when the rooms are placed next to each other. Each door is used as an 'anchor' to snap the rooms together.

Grouping rooms

To store and group rooms by type or theme, create a Level Room Group.

To do this:
1. Right click in the Project window
2. Select **Create** > **FPSKit** > **LevelRoomGroup**

Opening and placing room your first room

To open and place rooms for your level:

1. In the Hierarchy, select the **LevelLayout** gameObject.

2. In the Inspector, find the Level Layout component.

3. Select a Group from the available list to view all rooms in the group.

4. Select a room and move your cursor into the Scene view to preview room placement. The first room of your level will be fixed at (0,0,0) in relation to the GameObject on which the Level Layout component is placed.

5. Left-click to place the room.

Adding additional rooms

All additional rooms you add will connect to the exit closest to the cursor in the **Scene view**.

To add an additional room:

1. Move the mouse until the room is connected to the exit you want
2. Press R on your keyboard to rotate it and cycle through available exits
3. Left-click to place the room

Deleting rooms

1. In the LevelLayout component, select the Remove button
2. Hover over the room you want to delete
3. Left-click on it to remove it

The doors leading to that room will then close.

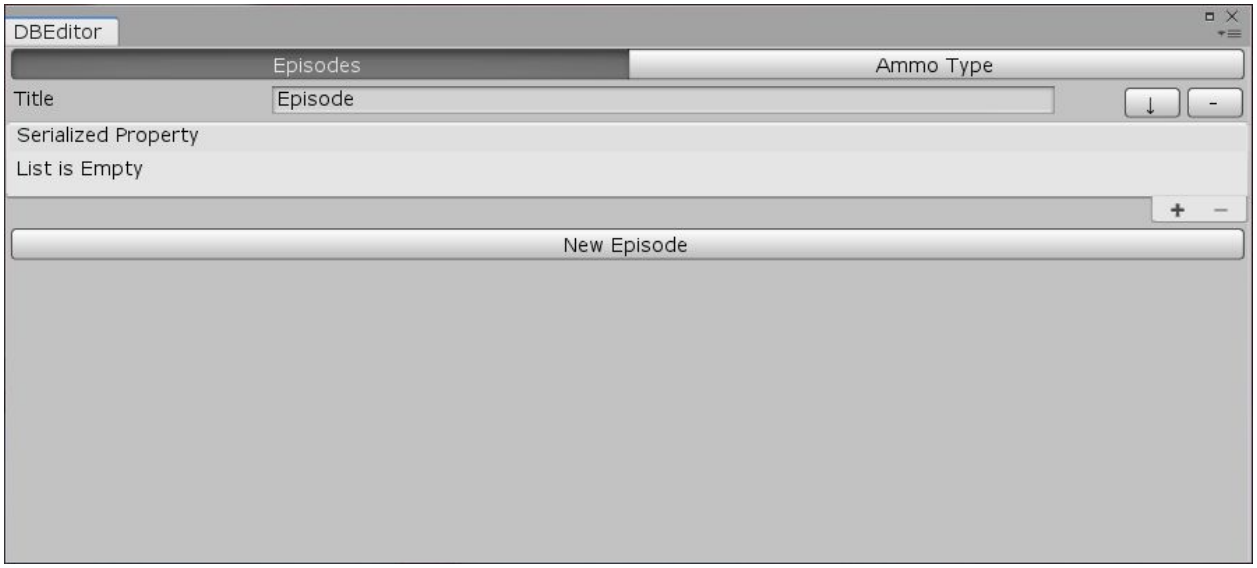
Episodes and Levels

In this Creator Kit, levels are grouped together into Episodes.

Creating an Episode

To create a new Episode:

1. Select **FPSKIT > GameDatabase** to open the Game Database
2. In the Game Database window, select the Episode tab and then New Episode
3. Type your Episode name into the Title field



Adding Scenes to an Episode

To add a Scene to an Episode:

1. Click the '+' button below your Episode Title
2. Drag and drop the the Scene into the Slot
3.

As you build the game, the scenes are automatically added to the build starting with the very first Level.

You can select a Level through the Select Level entry in the Pause Menu.

Minimap system

This kit offers a simple minimap system. It works by rendering some of the level mesh from a top down perspective inside a render texture that can then be displayed in the UI.

To know which mesh to render, the system relies on every mesh renderer that needs to be part of the minimap (typically the walls) to have a Minimap Element script attached to them. (See the room Prefabs in the kit for an example).

The system only renders active objects. You can even create specific meshes that are straighter and less complex than the level geometry. You can also put them in a layer rendered by no camera and add the Minimap Element script to them to create a more abstract Minimap.

The system also uses a given material to render the map. The default one renders all in white, but you could use a custom shader using vertex color on custom mesh to color code your map.

Note: This is a simplified system designed specifically for this kit. It doesn't have the optimization that a complete project needs. For example, it doesn't do any kind of pre-culling. Instead, the whole level is sent to the engine to be rendered and will only be culled by the viewport.

Adding a new weapon

All weapon models need to contain models of the Character's arms too as they'll be animated differently for each weaponCreating a new weapon from a 3D model

1. Import your 3D model with animations
2. Create and openan empty GameObject
3. Create a Weapon Script

Add your weapon model as a child object of the empty GameObject

1. Adding a weapon to a CharacterIn the Character Prefab, make the empty parent GameObject you created a child of the GameObject called WeaponPlace
2. Make sure its position is (0,0,0) in the Inspectorwindow

3. Move your weapon model until it's placed properly in the game view
4. Make the empty GameObject a Prefab and add it to a Character's weapon array

For the Animator Controller to give to your Animator on your Weapon model:

1. Create an Animation Override Controller by right clicking Create then selecting Animation Override Controller
2. Set WeaponBaseController as the Controller. This will give a list of clips called DummyFire and DummyReload.
3. Replace these animation clips with your own

Ammo

Defining Ammo typesTo define Ammo types, click the Ammo tab in the GameDatabase(**FPSKit > GameDatabase)**.

Ammo is just a name that allows different weapons to share one ammo type. If this is the case, the weapons will share the ammo from a common pool.

Placing Ammo Pickups

You can place Ammo Pickups throughout the game for Characters to collect torefill their ammo reserve.

To do this:

1. Create an Object with a Collider
2. Add an AmmoBox script
3. In the script, choose the parameter to determine which ammo type the Ammo Pickup will refill

The script will take care of setting the object in the right layer and change the collider in a trigger.

Lock and Door System

You can create doors that will only open if the player has collected the corresponding key.

You can either use an example door andkey (which can be found in the folder:

Creator Kit - FPS > Prefabs > Door and Key) or create your ownPlacing a door

The door Prefab is called FatBlob. To place the Prefab:

1. Drag it to the entrance you want to block
2. Click to place the door.

Placing a key

The key won't appear in the game automatically, you'll need to place it. If you don't, a warning message will appear telling you to add a key to the scene.

To place the key:

1. Drag the key Prefab from the folder to the desired location

Give it a name in the Key type setting in the Key componentLinking keys to doors

The name you entered in Key type setting will now appear in the Key type drop down in the Lock component. Select the name to link the key to the door it unlocks.Creating a locked door

To create your own locked door:

1. Add a Collider to the door (or a child of the door) and set it to trigger
2. Make it larger than the door so the player enters the trigger when they're close to the door
3. Add a Lock componentto the door (or child of the door)

The component contains a list of game actions (like a moving action that moves the door out of the way) that'll be activated when the player has the right key. For example, the example door has a dissolve action that animates a parameter of the material to make a dissolve effect. In the Lock component there is also a dropdown for you to select the Key type you want to open the door.

To create a key, add the Key Component to the door.

Creating a new level

To make a new level, you need to create a new Scene.

Duplicating the example SceneTo create a new Scene that has the same look as the example:

1. Click the Create New Scene button in the FPSKIT menu. A file explorer window will appear.
2. Name and save your new Scene

The new scene will contain all the basic objects you need, including an editable LevelLayout.
Creating a Scene from scratch
If you want to create a new scene from scratch, it should:

- Contain a Character Prefab
- Not use a camera (The Character Prefab already contains a camera. If you want to change FOV or post process, duplicate the Character Prefab and modify its camera. Use this new Prefab instead.)
- Have a GameObject with a GameSystem script on it. The script contains a list of Start Prefabs that it'll instantiate when the game starts. You usually want at least the GameUI Prefab in there (because lots of code from the controller interacts with it) and an ImpactManager (to handle the pooling and spawning impact effect when shooting).

Design notes

This section:

- highlights some design decisions
- explains them for reference, and
- highlights any alternatives or improvements.

Please remember that the goal of this project is to offer a simple, easy to modify and easy to extend system. Some design decisions are not optimal for a full game and could reduce performance. We've highlighted decisions where this is the case.

Weapon rendering

The weapon rendering uses a second camera with a smaller Field of View (FOV) of 40 degrees. This is because:

- You can render weapons on top of everything else so they don't clip through walls or other objects when the player gets close to them
- A smaller FOV makes the weapon feel closer and not look distorted on larger FOV
- Everything can be rendered in relation to the weapon (for example, particle effects and Muzzle flash)

This does, however, come with two significant drawbacks:

- The weapons won't have shadows (though they'll still receive lighting data from probes)
- Postprocessing on the main camera won't affect it, so we have to do two postprocess profiles:
    - one that affects depth (like depth of field) since depth is cleared by the second camera, and

- one that affects colors (like color grading) on the weapon camera

There are other ways of doing this, like:

- rendering the weapons very small and close to the camera, or
- using special shaders on the gun to render it at specific FOV and above everything.

But these methods also have multiple drawbacks too.

We chose the multiple camera approach as it offers the most ‘plug and play’ solution that’s acceptable for a kit or prototype project.

Impact Manager

The Impact Manager is a script that handles the spawning and impact effect when a raycast from a weapon hits a surface.

The implementation in the FPS Kit means you can:

- Define a default sound and particle system to use on impact
- Use a list of overrides that allow you to define a specific sound and particle effect attached to a material. When the game starts, it builds a dictionary (for faster lookup) matching a material to an override.

When a ray hits something, check if:

- there’s a renderer on the collider or one of its children
- there’s an override for the material that renderer uses

This means that you don’t have to edit all the objects that can be hit by a raycast, but the performance does take a hit because of the renderer lookup.

For a full game, you could do one of the following instead to maintain performance:

- Use the Physic Material on the collider instead of a rendering material. This is faster (as there’s no need for a lookup and you already have the collider part of the raycast result) You do have to manually set the material on all your collider which requires more setup (but you might need to do that in a full game anyway). The different Physic Materials can also be used by other raycasts (like the grounded one) to play different footsteps on the surface

- If you need to handle something more specific like finding which submesh is hit, add a script on every hittable object that registers the object collider in a dictionary for a faster lookup from collider to renderer

Components reference

This section lists the parameters of all the components added by the Kit.

Controller

The Controller processes player input and controls the Character. There's a Prefab available in the Prefab folder called Character.

Controller settings

- **Camera Position**: A Transform that's either the object on which the main camera is or a parent of it. This is the Transform that's rotated to look up and down.
- **Weapon Position**: A Transform under which the weapon will be placed. You want it to be under the Camera Position so the weapon moves with the camera.
- **Starting Weapons**: An array of Weapons with which the player will start.
- **Starting Ammo**: A list of ammo types and the amount with which the player starts the Level.
- **Mouse Sensitivity**: Allows you to tweak how much the camera moves with Mouse movement. Higher values mean faster tracking.
- **Player Speed**: The speed at which the character moves (in units per second).
- **Jump Speed**: The vertical speed given to the character when they press the jump button (in units per second). Higher values mean higher jumps.

Weapon Settings

- **Trigger Type**:
  - **Manual**: The player needs to release the mouse button to fire the weapon again
  - **Auto**: The weapon will fire as long as the button is pressed (based on the fire rate value)
- **Weapon Type:**

    - **Raycast**: Each shot is raycast and the target is considered hit instantly if a raycast touches it (fast pellets and lasers, for example)
    - **Projectile**: Each shot is instantiated and physically launches a projectile. The projectile usually has a separate script that handles its behaviour (explode and hit target on contact, explode after a while, or is propelled forward, for example)
- **Fire Rate**: How long between each shot. In manual mode, pressing the trigger twice faster than the fire rate means the second press is ignored. In auto mode, it's the time between two shots when the button is being pressed.
  Note: The fire animation time is scaled to fit the default fire rate. It's useful to be able to make small tweaks to the fire rate, but changing it too much can result in weird animation effects.
- **Reload Time**: The time it takes to reload the weapon.
  As above, the reload animation is scaled to fit the default reload time. We recommend only adjusting it by +/-1s.
- **Clip Size**: How much ammo there is in a clip. Once the clip's empty, you can't fire anymore until you reload.
- **Damage**: How much damage each hit does. In cases of multiple projectiles per shot (see Advanced Settings below), each projectile does that amount of damage (for example, a weapon with damage set to 1 and shooting 4 projectiles will do 4 damage if all 4 hit the target).
- **Ammo Type**: Which ammo each weapon uses. This allows multiple weapons to share a pool of ammo. Use the Game Database to create new Ammo Type.
- **Projectile Prefab** (only for Projectile-type weapons): The Prefab of the projectile each weapon uses.
- **Projectile Launch Force** (only for Projectile-type weapons): The force used to launch the projectile.
- **End Point**: A Transform placed at the Weapon End to mark where effects and launching the projectile should start.
- **Advanced Settings**: Setting to personalise more weapons.
    - **Spread Angle**: A cone that defines an area in which the raycast or launch direction will be randomly picked. The larger the cone, the less precision the weapon has.
    - **Projectile Per Shot**: How many raycasts or projectiles are fired with each shot. Note that each will have a random direction picked within the Spread cone.
    - **Screen Shake Multiplier**: Used to increase (or decrease) the screen shaking effect for each shot. The default value is 1.
- **Animations clips**: The weapon requires a reference to the fire and reload clip so it can scale the time based on the fire rate and reload setting.
- **Audio Clip**: The various audio clips to play at a given event

- **Prefab Ray Trail** (Raycast only weapon)*:* Prefab of the Line render used to visually trace the Raycast. If none is given it will be invisible
- **Visual Display**: This references a Component of AmmoDisplay type that will be notified when the ammo amount changes (fire or reload). The kit offers two default implementations of TextAmmoDisplay that can change a UI text, and a LiquidAmmoDisplay that changes the level of liquid in an object using the LiquidContainer component (see kit Weapons for samples).