photon

Search

This document is about: **QUANTUM 2**

SWITCH TO

Entity View

Introduction

The `EntityViewUpdater` and `EntityView` are responsible for handling the view side of all entities, in Unity. These Unity components performs logics such as destroying, creating and updating the view game objects for entities, based on data from the simulation. The `EntityViewUpdater` is a `MonoBehavoiur` that needs to be present in all game scenes which contains a Map.
While the built-in implementations are good enough for prototyping and simple use cases, in many cases a custom implementation is needed. Many functions on the `EntityViewUpdater` and `EntityView` are virtual and can be overridden. Alternatively the implementations can be completely replaced with a custom implementation. The sources for the `EntityViewUpdater` and `EntityView` are available in the Unity project.
`EntityViews` are linked to `Entities` via the `View` Quantum component which contains an `AssetRef` to the view GameObject to spawn for the entity. When configuring an Entity Prototype prefab or scene object with an `EntityView` in Unity, the `View` component is automatically saved into the prototype with the correct view.

Bind Behaviour

Entity View components in Unity have a field called Bind Behaviour which can be set to either:

- `Non Verified` : the view Game Object can be created in Predicted frames;
- `Verified` : the view Game Object can only be created in Verified frames;

Using `Non Verified` is usually better for the views of entities which are instantiated in high frequency and/or they need to show up as quickly as possible on the player screen due to gameplay reaction time mechanics and such. For example, **creating projectiles in a fast paced shooting game** should be done using this alternative.

show up immediately and can afford the small delay of waiting for a Verified frame. This can be useful to avoid creating/destructing view objects during mispredictions. A good example of when to use this is for the **creation of playable character entities**. This avoids some issues such as when configuring player entity views, which could lead to issues if created and reallocated in `Non Verified` mispredictions.

EntityView Pooling

By default, the `EntityViewUpdater` creates new instances of the `EntityView` prefabs whenever an entity gets created and destroys the view GameObjects respectively when an entity gets destroyed. This can be quite CPU expensive especially on mobile platforms. Manual pooling of the views can be used to pool the GameObjects instead and improve performance.

Overriding Create

To get the GameObjects from a pool instead of having them instantiated, override the `CreateEntityViewInstance` function. The function has an `EntityViewAsset` parameter indicating which view to spawn. The `EntityView.AssetGuid` can be used as a key in a dictionary of pooled objects.
Example implementation

C#

```
protected override EntityView CreateEntityViewInstance(EntityView
    Debug.Assert(asset.View != null);

    EntityView view = _myObjectPool.GetInstance(asset);

    view.transform.position = position ?? default;
    view.transform.rotation = rotation ?? Quaternion.identity;

    return view;
}
```

cases this is not necessary. When overriding it is important to keep the EntityRef assignment in place.

Overriding Destroy

To return views to the pool instead of destroying them you can override `DestroyEntityViewInstance`.
Example implementation

C#

```
protected virtual void DestroyEntityViewInstance(EntityView insta
    _myObjectPool.ReturnInstance(instance);
}
```

Map Entities

For map entities `ActivateMapEntityInstance` is responsible for activating the views and can be overridden for custom behavior if needed.

`DisableMapEntityInstance` gets called which by default disables the GameObject. This function can be overridden for custom behavior as well.

Manual Disposal

`EntityView's` have a `Manual Disposal` property that can be toggled in the inspector. When enabled the destruction methods in the `EntityViewUpdater` are skipped. This allows for manual destruction using the `OnEntityDestroyed` callback of the `EntityView` or to destroy them via custom destroy events.

Finding EntityViews

A very common use case is to find the view of a specific entity. Since the simulation side is not aware of `EntityViews`, `EntityRefs` must be passed to the view via events. The

photon

Events and EntityView Update Order

The `OnUpdateView` function on the `EntityViewUpdater`, which is responsible for creating, destroying and updating `EntityViews`, gets called before events get processed. This means that destroyed entities in an event might already had their views destroyed.

Custom Destroy Events

A common pattern is to destroy and entity but still wanting to execute an event with additional information about the destruction to the view. To prevent the `EntityView` from getting destroyed before the event gets processed set `Manual Disposal` to true on the `EntityView`.
This will keep the `EntityView` alive instead of passing it into the `EntityViewUpdater's DestroyEntityViewInstance` function which by default destroys the GameObject.
With that the event handler can still find the view and execute the destroy event with the view present. The `EntityView` needs to be cleaned up manually by destroying it or returning it to the object pool.

AutoFindMapData

AutoFindMapData has to be enabled when using maps with `EntityViews` on them. If enabled the view will search for the corresponding MapData object and match map entities with their views. Disable this if you are not using maps with entities to allow for having scenes without a `MapData` component present.

Custom Interpolation & Teleporting Entities

The `EntityView` component interpolates the entity `GameObject` visuals by default. This is to adjust for the difference in simulation rate and render (update) rate and for error correction in terms of mis-prediction.

override the `ApplyTransform` function. The `UpdatePostionParameter param` contains all the information necessary for displaying the entity. Use `param.UninterpolatedPosition` and `param.UninterpolatedRotation` to implement custom interpolation and snap to the uninterpolated position during teleports to display the visuals correctly.
Additionally, for resetting the interpolation of every entity there is `TeleportAllEntities` on the `EntityViewUpdater` that can be used when resetting the map to deactivate interpolation for all entities for a frame.

Back to top