AAAPE Dots Utilities

Installation & Setup

In Order to install AAAPE Dots Utilities into your current project please follow these easy steps:

1. Head over to https://github.com/aaape-games/unity-dots-utilites/releases
2. Download the newest release by clicking on the *.unitypackage
3. Drag the unity-package into your unity-editor - project files window

Dependencies

Unity Editor Version

Unity entites currently supports 2020 LTS and updwards, meaning your Editor must be at least version 2020.3.31 to run enitites

Package Dependencies

If you have installed this package via package manager, you can skip the dependencies, because they're installed for you by unity.
If you downloaded from github, please download this packages from the package manager manually:

- com.unity.entities:0.17.0.preview42 upwards

Installing Entities

If you want to install Unity entities in a specific version, there are 2 ways depending on which editor version you run.

- Unity 2020.x : go to the package manager windows click on the + button and choose "install from github url", enter "com.unity.entities" and submit.
  This will download you the newest version of Entities (currently 0.50). If you want to go with Entities 0.17.0, you need to manually go into the Packages/manifest and

adjust the version to “0.17.0-preview.42”. When Moving Back to the editor, it will automatically get the version you entered.
- Unity 2021.x: open up the package manager and choose “add package by name…” here you enter into the name box “com.unity.entities” and into the second box the version you want to download “0.17.0-preview.42” make sure to use the full version number. Note that Entities 0.50 is not supported for Unity 2021 yet.

URP

When you are using Entities 0.50 you are bound to URP. The built-in (legacy) render pipeline is not compatible with Entities 0.50+.
The otherway round 0.17 is somewhat buggy when it comes to URP (remember, its an early-access preview package).
All the demos are made with entites 0.17 so you can either stick with that or simply upgrade all the materials

Unity 2021.x

- Make sure to backup into your VCS.
- Window -> Rendering ->Render Pipeline Converter
- Choose “Built-in to URP”
- Check “Material Upgrade” and “Readonly Material Converter” (the boxes are a little hard to click, make sure you actually checked them.
- Click “initialize Converters”
- Click “Convert Assets”, this can take quite a while, especially when adding Readonly Material Converter and you are having a big project

Unity 2020.3.x

- Make sure to backup into your VCS.
- Edit -> Render Pipeline -> Universal Render Pipeline -> Upgrade Project Materials to UniversalRP Materials

Assembly Definition Files

If you’re using Assembly definition files please add the following references
- aaape.dots.runtime

Features

Gamestate

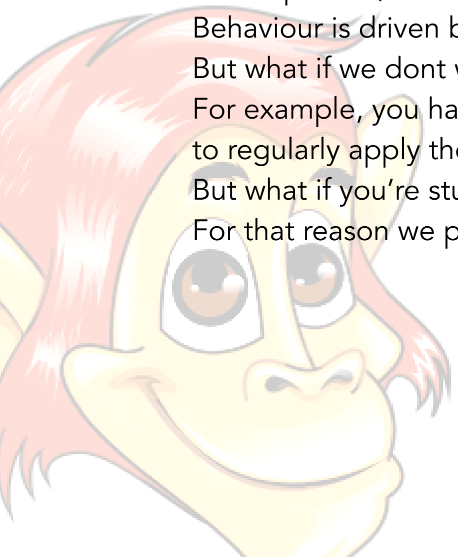
First of all, what is a Gamestate and when to use?
GameStates are basically flags that are present for all systems, they can be used to store a “state” of a game and how “everything” (we’ll come to that later) should behave when a certain precondition is met.
A good example is a “pause”-state, in most games you will just make the level stop, you cant move your character etc.; Maybe you want to continue to play music.
Maybe your enemies can still move with super-slowmo?
You can handle all this with a gamestate.

Here is a quick example from the States-Demo.
Whenever the Gamestate-Flag “NoGravity” is set, all characters will start moving upwards slowly. When the flag is removed, something different can happen.
How AAAPE Dots handling this is to “RequireForUpdate”, meaning in the onCreate method of a Systembase, you define if the OnUpdate Message should only be toggled if this Flag is present, only then running the filter and executing it, which is very performant.
You could also use the ECS-inbuilt “RequireSingletinForUpdate<Generic>”, but AAAPE will in the future provide more functionality for those State-changes, like eventhandlers etc.

```
public class GuyLiftSystem : SystemBase
    {
        protected override void OnCreate()
        {
            // this is a gamestate
            GameState.RequireForUpdate<NoGravityFlag>(this);
            // you could also use
            // RequireSingletonForUpdate<LevelStartedFlag>();
            // which is doing the same
        }
```

Stateful Components

Once again, what is a Stateful Component?
A Component, in Entities Language is more a less Data. Data is nothing without Behaviour.
Behaviour is driven by the Entities “Systems”.
But what if we dont want to run Behaviour on every entity containing this Component?
For example, you have a Unit that has an MovespeedComponent and a Target, do you want to regularly apply the damage to the target? Yes - right.
But what if you’re stunned by a hammer-attack? You can’t attack.
For that reason we provide you with Stateful Components.



A stateful Component always contains at least one state.
Back to our Unit example, it has Following States “Idle, Attack, Stunned”, we only want to run a system (Behaviour) when the “attack” state is present.
Here’s how:

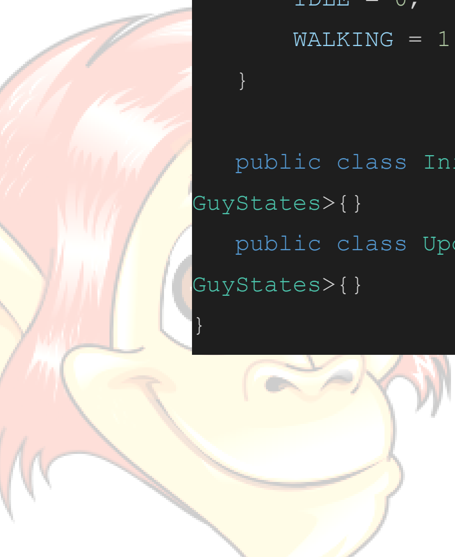
```
protected override void OnUpdate()
       {
           Entities
               .WithAll<Unit>()
               // here you can set to only run action on components with state x
               .WithSharedComponentFilter(new UnitState{Value =
UnitStates.WALKING}).ForEach(....
```

What do we have here?
In the OnUpdate Method of a system, we tell entities to query for all Units that do have a SharedComponentFilter, our actual state and run the action only on those.

Basic Implementation

The Basic Implementation looks a little overwhelming at first, but its pretty easy as you get a hang of it

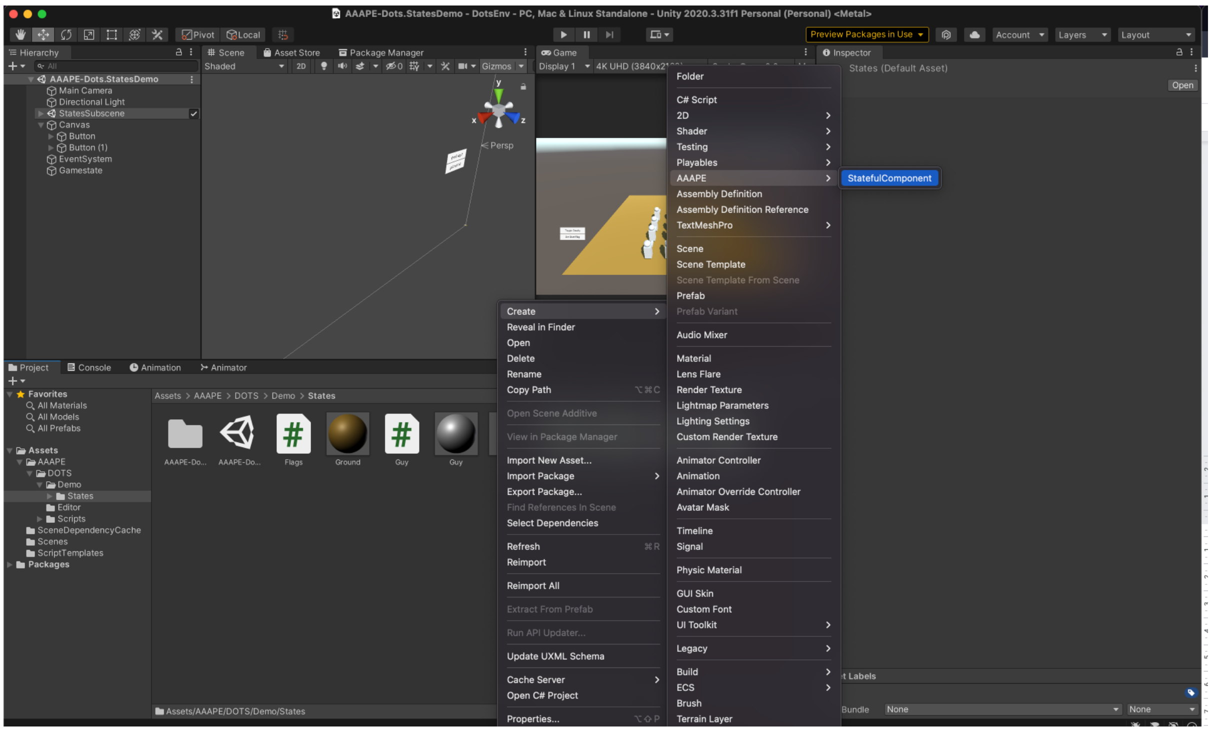
```
  [GenerateAuthoringComponent]
   public struct Guy : StatefulComponent<GuyState, GuyStates>
   {
       public GuyState state;
       GuyState StatefulComponent<GuyState, GuyStates>.State { get => state; set =>
state =value; }
   }

   [System.Serializable]
   public struct GuyState: State<GuyStates>{
       public GuyStates Value;
   }

   public enum GuyStates {
       IDLE = 0,
       WALKING = 1
   }

   public class InitGuyStateSystem: InitStatefulComponentSystem<Guy, GuyState,
GuyStates>{}
   public class UpdateGuyStateSystem: UpdateStatefulComponentSystem<Guy, GuyState,
GuyStates>{}
}
```



- Guy: The Guy is the StatefulComponent, holding and delivering the state, in form of a SharedComponentData-State
- GuyState: GuyState is the actual StateData
- GuyStates: GuyStates is the enum, where we can define all of our States, if you want to add a “stunned” stated, do it here.#
- the other two systems are convenient systems that sync the Guy - State with the GuyState.

But, you dont need to write everything on your own.
AAAPE Dots brings a scripttemplate that generates you everything for your project on the fly.

- Go into the Unity Inspector and right clcik into the proejct-window
- Create->AAAPE -> Stateful Component



- Enter the Name of the Component , e.g. “Guy” or “Unit” and press Enter

Unity will create everything for you.
You can now add the Generated Component to your Entity(Prefab) or (conversion)-Gameobject and select the default state. Thats it!