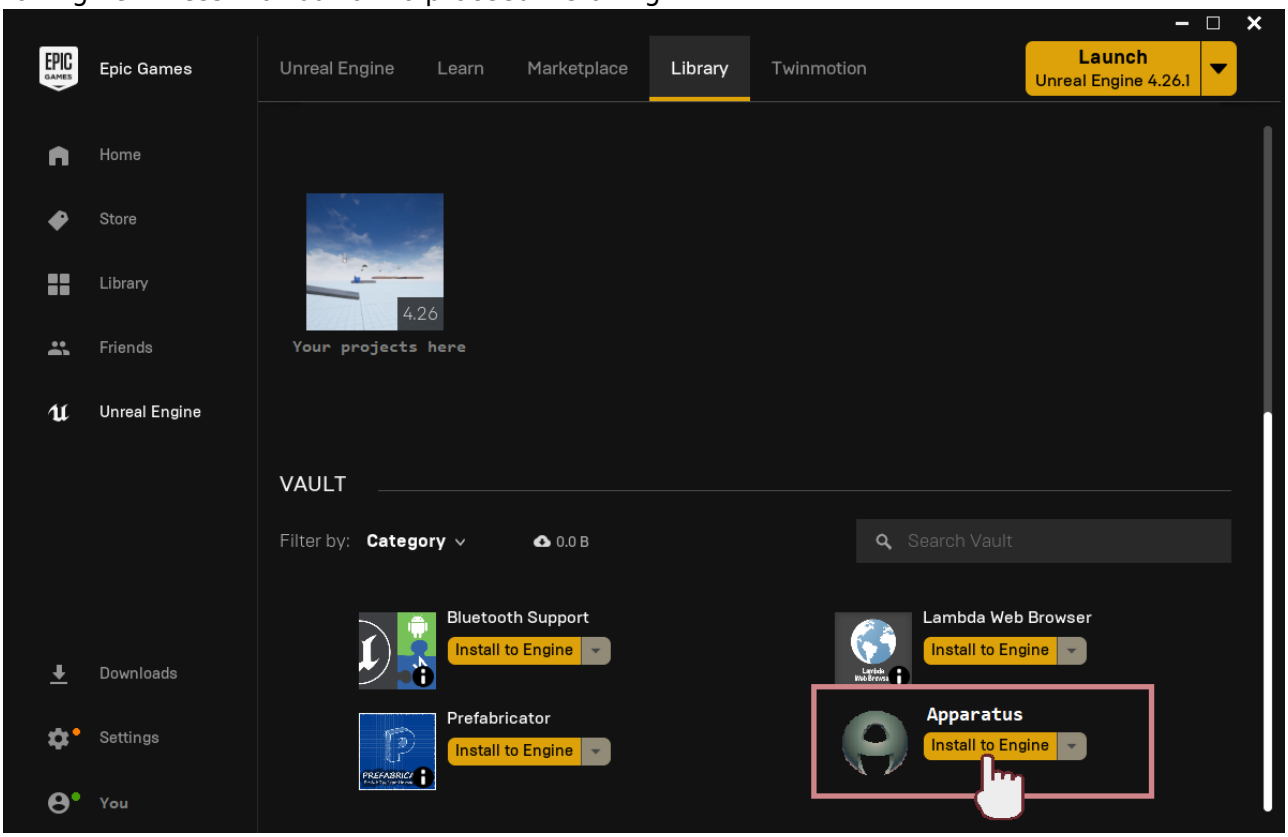
# Apparatus: Beginner's Guide

In this prolonged tutorial we will talk about using the Apparatus plugin inside Unreal Engine. You will create your first detail and implement the game mechanics in a special Blueprint class. We will demonstrate the most important features on the example of a simple two-dimensional platformer.

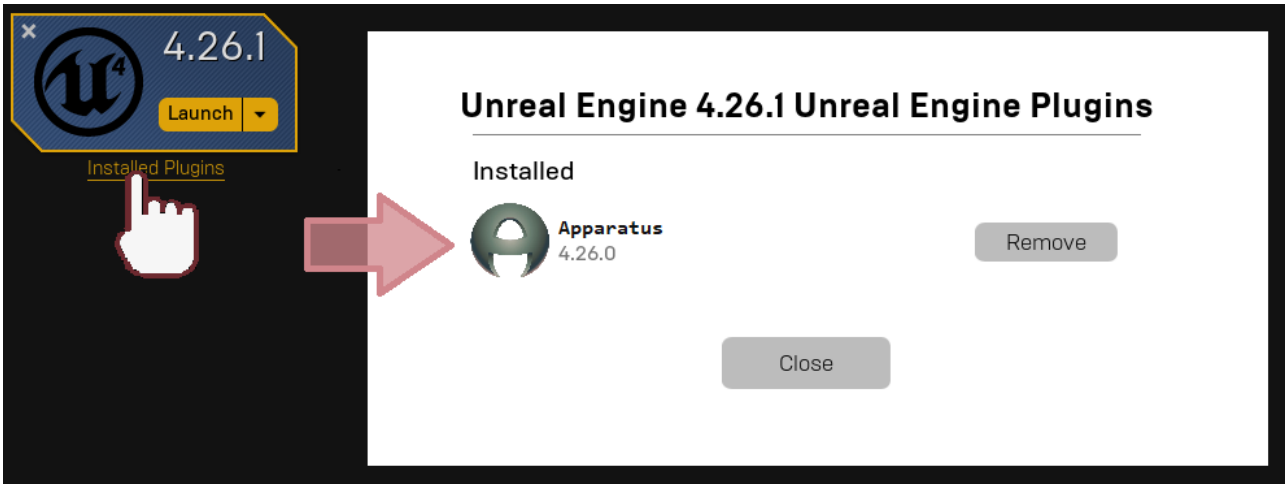
Just before going any further, make sure you have a basic understanding of ECS. Check our brief introduction to the ECS concepts.

## Plugin Installation (Activation)

1. Before creating a new project, you have to add the Apparatus plugin to the Engine. In order to do that, please, launch 'Epic Game Launcher' and make sure that 'Unreal Engine' item is currently selected on the left-most menu. Then navigate to the 'Library' tab on the top. Scroll down to the 'Vault' section to find the Apparatus plugin along with a yellow button titled 'Install to Engine'. Press that button to proceed installing.

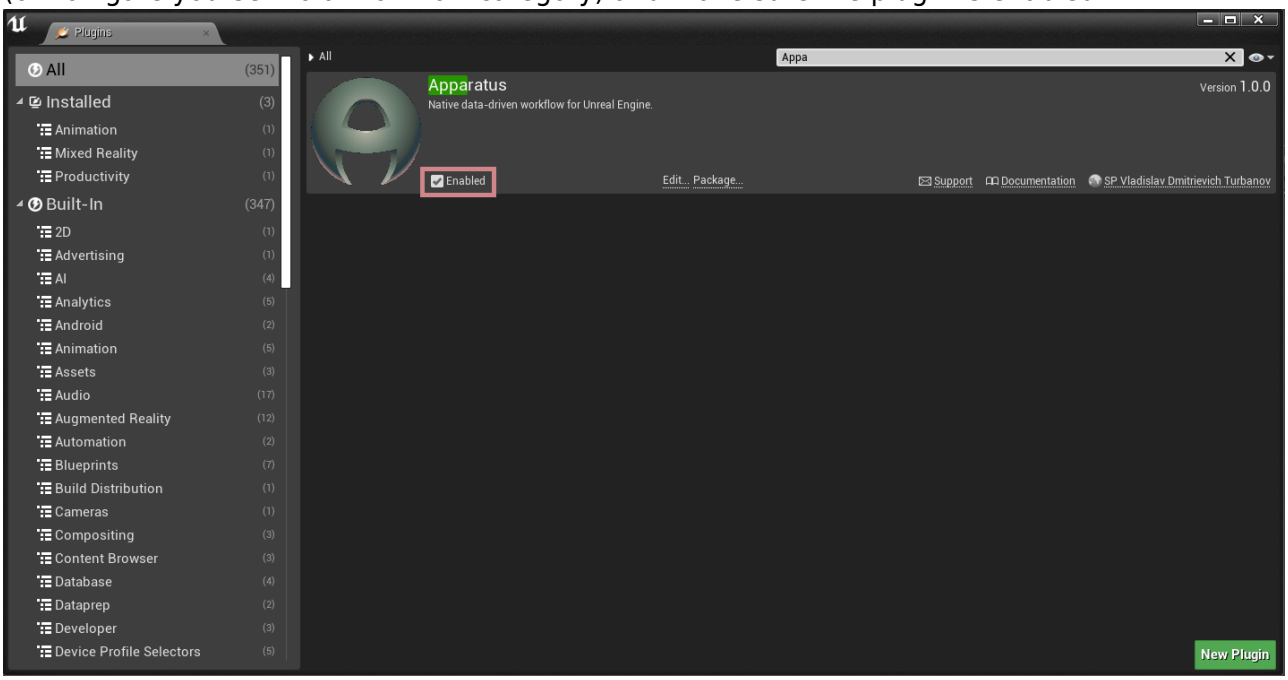



1. In the opened section choose the version of Unreal Engine. Note that you should use 4.26.1 (or above) as it's the only officially supported for now. After clicking 'Install' wait for a minute while the launcher loads the necessary files to the engine. When the installation has completed, you can verify its success by clicking the 'Installed Plugins' footer under Unreal Engine version you have installed the plugin for.



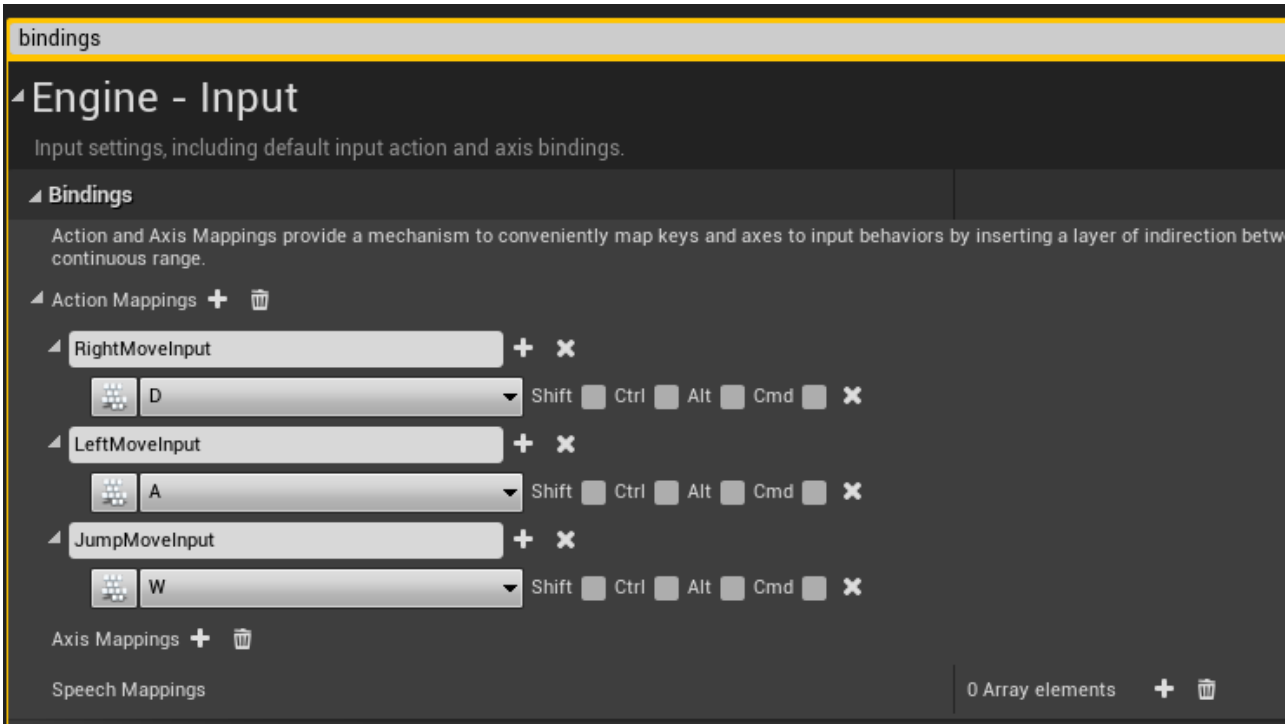

## Project Creation & Initialization

1. Now you should Create New Project as usually. Select the blank template, cause we will be building from scratch. Leave the other settings with their defaults. We will name the new project 'ApparatusLearn', but you can make up your own title if you want.
2. After the project is created and opened, you can check if the plugin is currently in use: in the uppermost main menu panel choose 'Edit' → 'Plugins'. Then type 'Apparatus' in the search box (or navigate yourself to a Workflow category) and make sure the plugin is enabled:

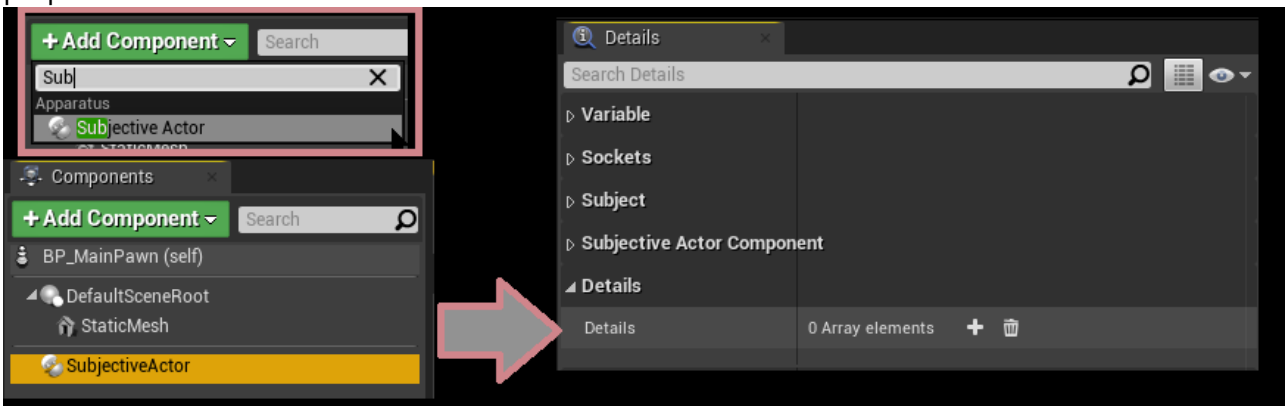



## Implementation

1. Ok, now. First of all we need to create a key binding so we can understand, when we should add the necessary details to the actor. In order to do that, go to 'Edit'→'Project Settings' and type 'bindings'. Focus on the list of 'Action Mappings' and add the following keys:
   - 'RightMoveInput' – keyboard key D
   - 'LeftMoveInput' – keyboard key Alt
   - 'JumpMoveInput' – keyboard key W
2. The result must be something like that:

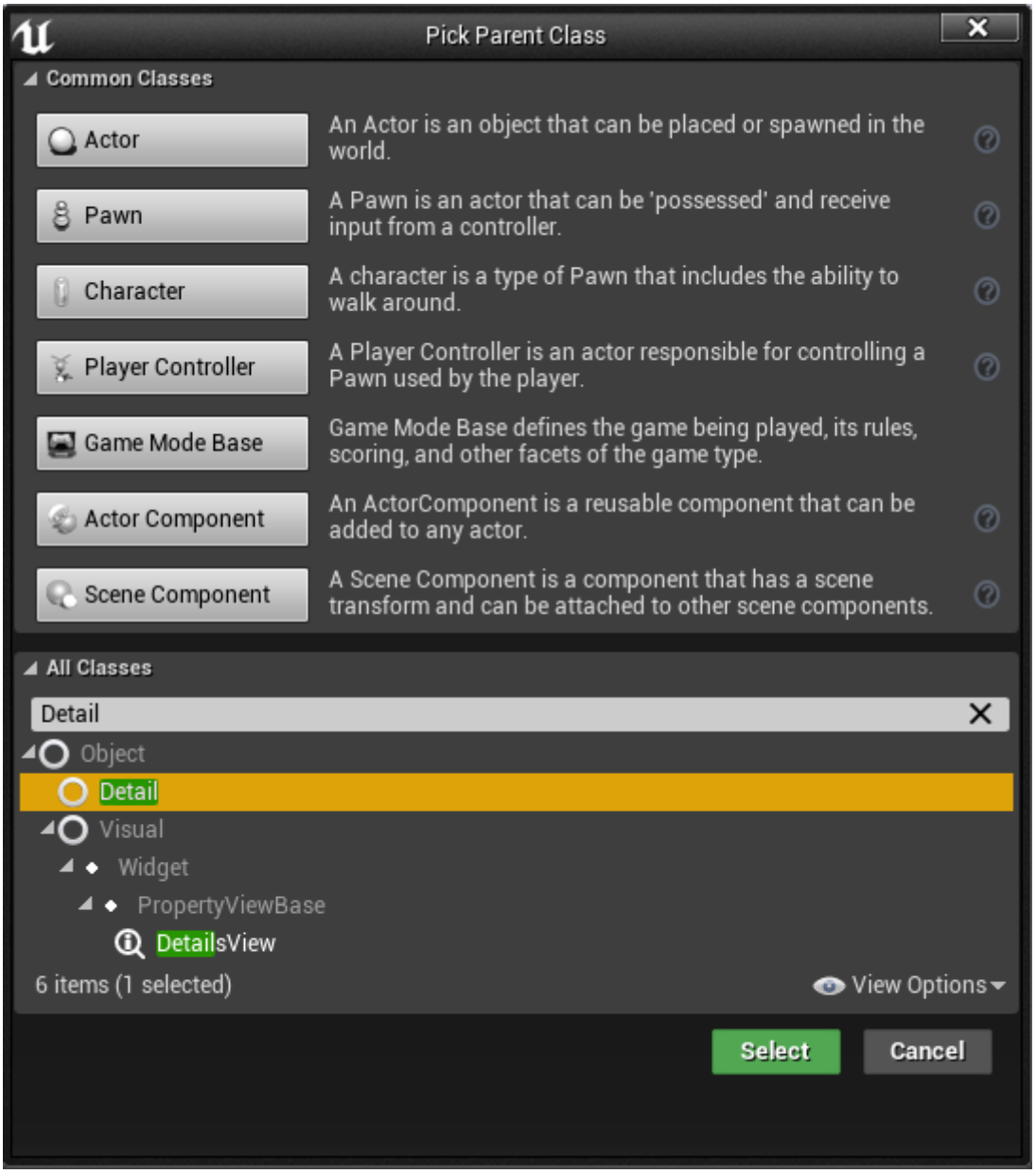



3. We proceed with creating a new Pawn Blueprint. In order to do that click green button 'Add/Import' in the 'Content Browser' and select 'Blueprint Class'→'Pawn' and give it a name 'BP_MainPawn' (for a sane naming convention practices, please, check this style guide). While selecting the Blueprint in the 'Content Browser' press Ctrl+Shift to save the newly created item. Now open it by double-clicking on its thumbnail (if you are really new to Content Browser, we're advising you to read the official Manual). Navigate to 'Event-graph' and delete all the nodes by Ctrl+Alt and Del. You should also understand how to use Blueprint Editor; hereinafter BP).
4. In BP Editor go to viewport and in the 'Components' section add 'StaticMesh'. In the 'Details' panel on the right pick the 'Cube' model for the 'Static Mesh' property. For our pawn to look prettier, choose the 'BrushedMetal' material in the 'Element 0' property. Add a 'Subjective Actor' component (provided by the Apparatus plugin) to the Pawn and see the 'Details' panel to become more familiar with the new Actor Component. In this guide we will only use the properties under the 'Details' section:

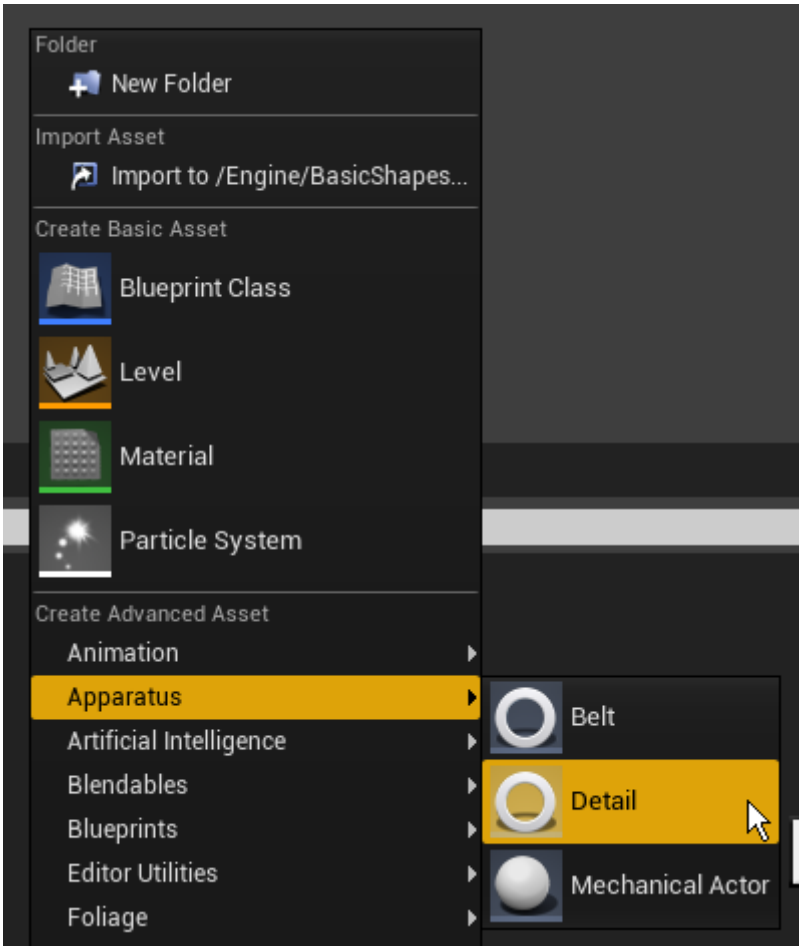



As you may notice, we will add our details there.

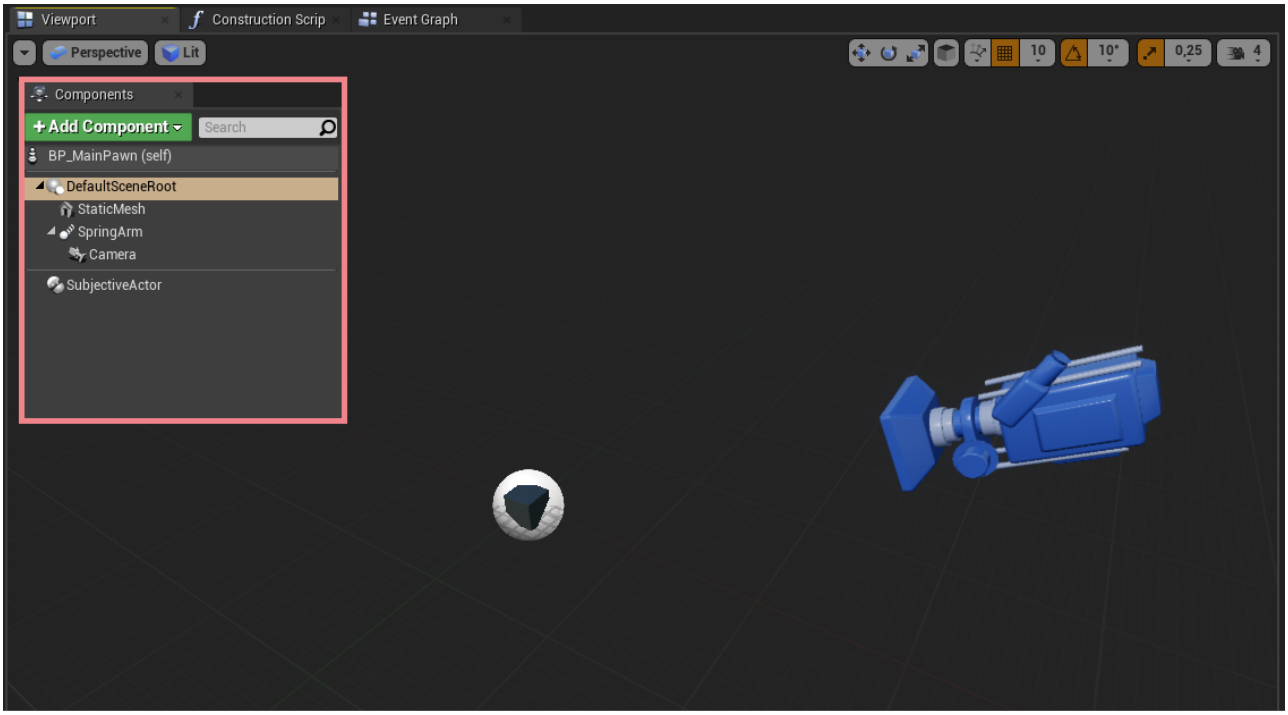
5. Ctrl+Shift+Shift to save everything, and compile the BP. Once again open the Content Browser, create a new BP but now expand title 'All classes' and in the hierarchy find 'Detail':



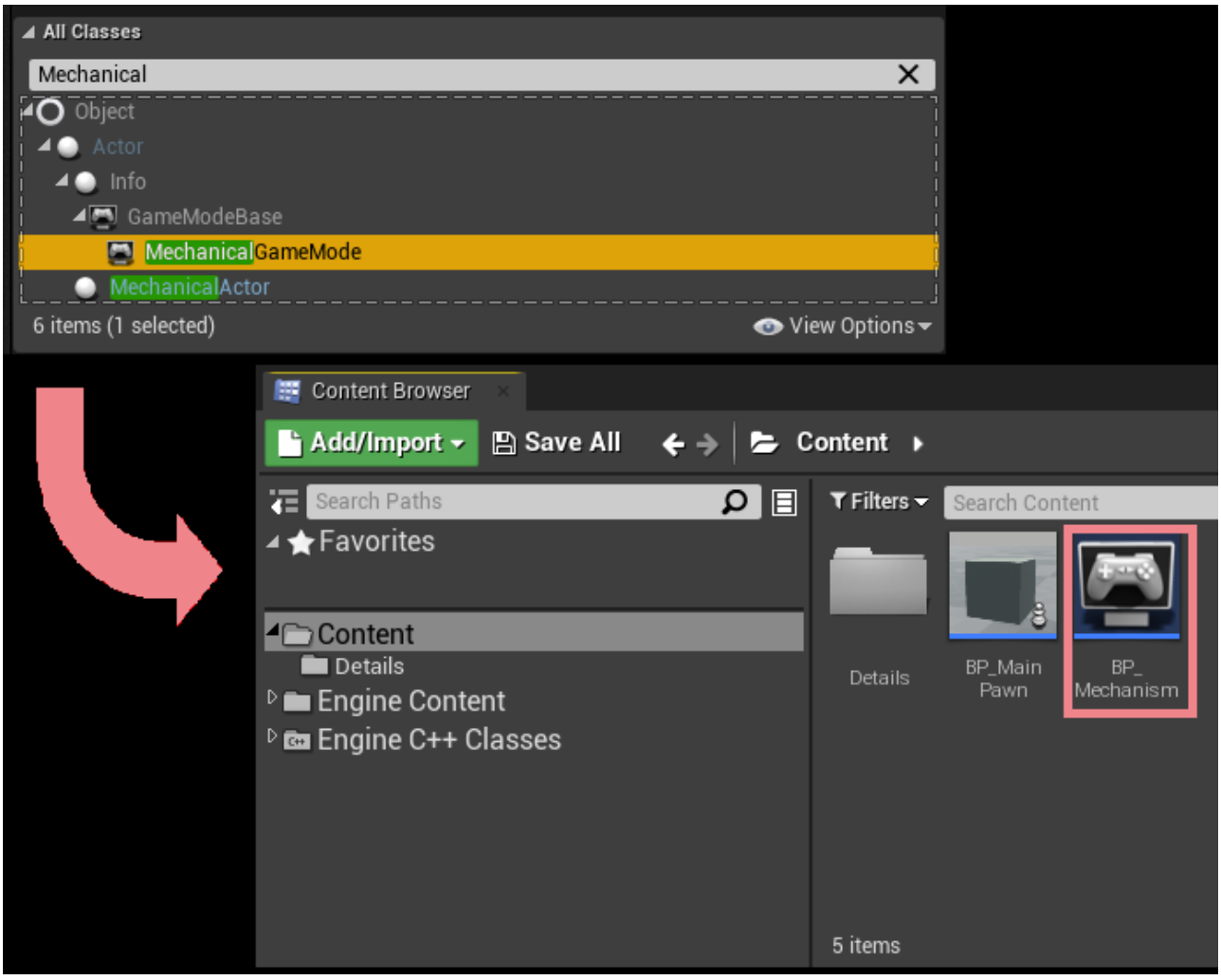

If you need to create a new Detail you can also use the ‘Advanced asset’ section of content browser (using the right mouse button)



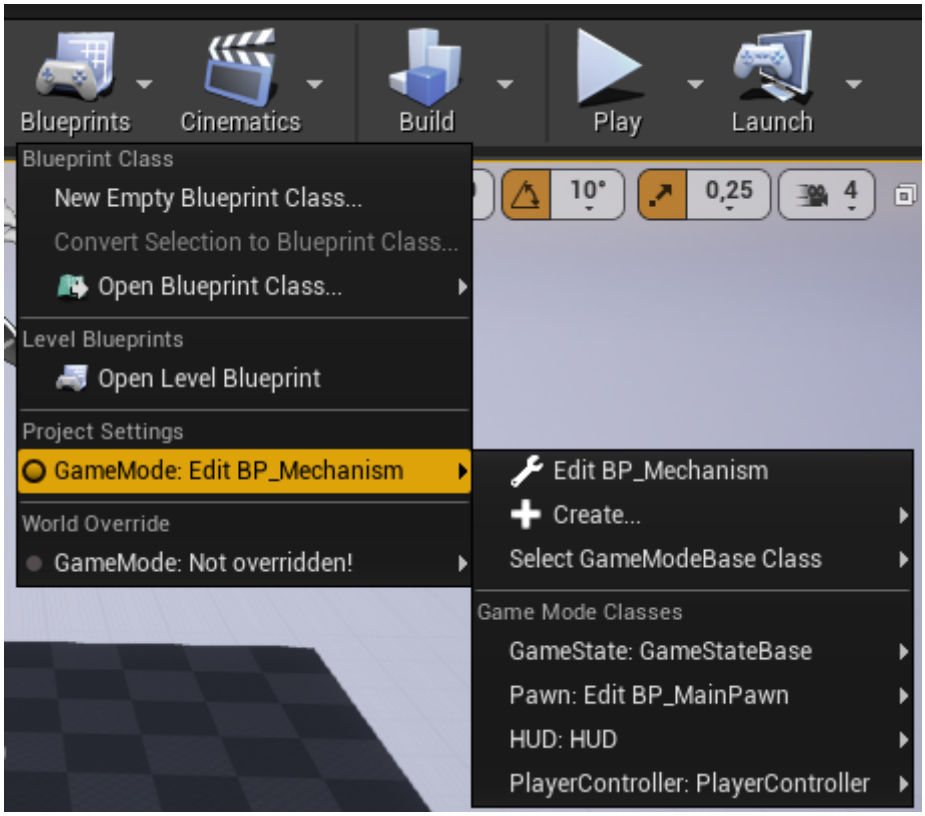

6. Generally speaking, you can and shoudl create as many details as you want, but for the tutorial sake we need specifically the following:
   - `D_Moveable`,
   - `D_Moving`,
   - `D_OnFloor`,
   - `D_Fallable`.
7. Keep your project organized and move all the Details classes to some folder called 'Details'. Open any detail you want in the BP Editor. As you can see, Apparatus' detail is actually a usual Blueprint class, inside which you can declare variables, macros and functions as you please. There are also two events that you can override ('Activated' & 'Deactivated') which are called when the detail's Boolean state 'Enabled' is set to either true or false. In the corresponding details add next variables:
   - Float 'Speed' to **'D_Moveable' class** with a default value of 100.0
   - Editable enum 'ETouch Swipe Direction' with the name 'Direction' to **'D_Moving' class**
   - Editable & expose on spawn Box Collision Object Reference with 'Bottom' name to **'D_Fallable'**
8. Go back to the 'BP_MainPawn' and make the cube a little smaller (for example 0.25,0.25.0.25 vector scale). Add 'Spring Arm' component to the 'DefaultSceneRoot' and the Camera to the arm (make sure that transform vectors are set to their default values); then firstly rotate the arm over Z axis by 180° and after that rotate it over Y axis by -30°.
9. You should get something like that:



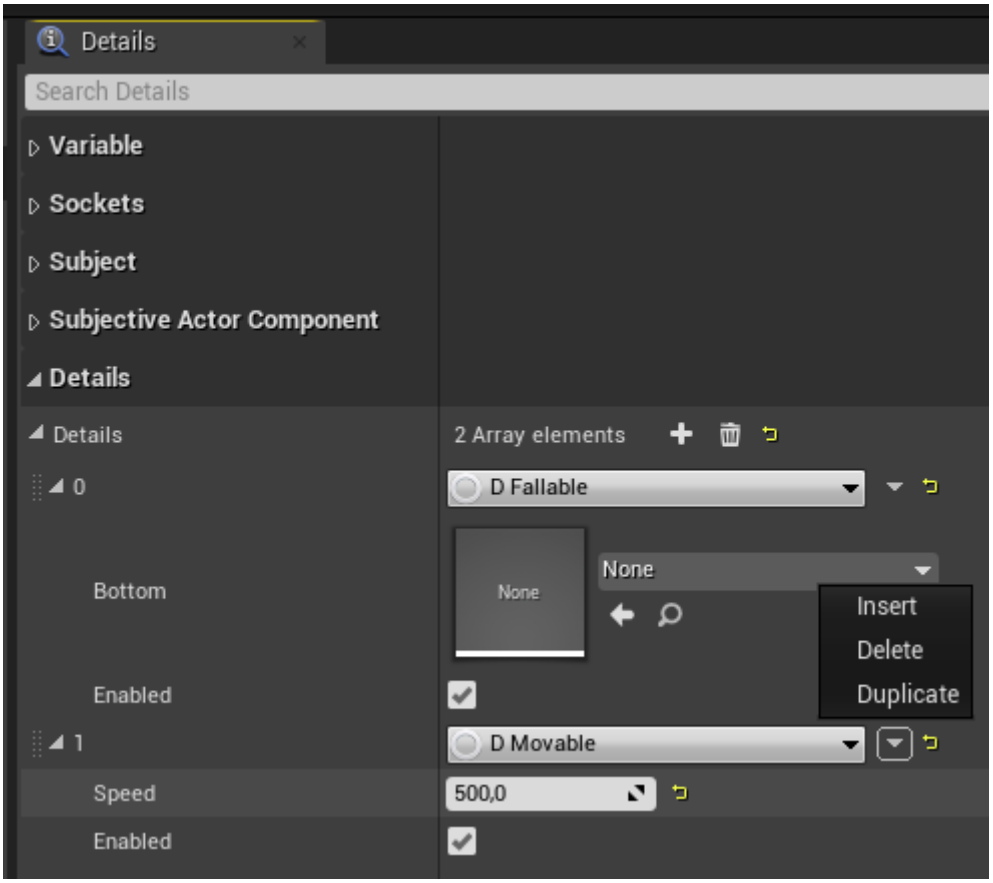

10. Create ‘GameMode’ class inherited from ‘MechanicalGameMode’ and name it ‘BP_Mechanism’. Like that:




11. Open ‘BP_Mechanism’ in BP Editor and in Details panel set ‘Default pawn class’ to ‘BP_MainPawn’. Go to the level settings: ‘Blueprints’→‘Project Settings : GameMode’ and select ‘BP_Mechanism’ as project ‘GameMode’ class:



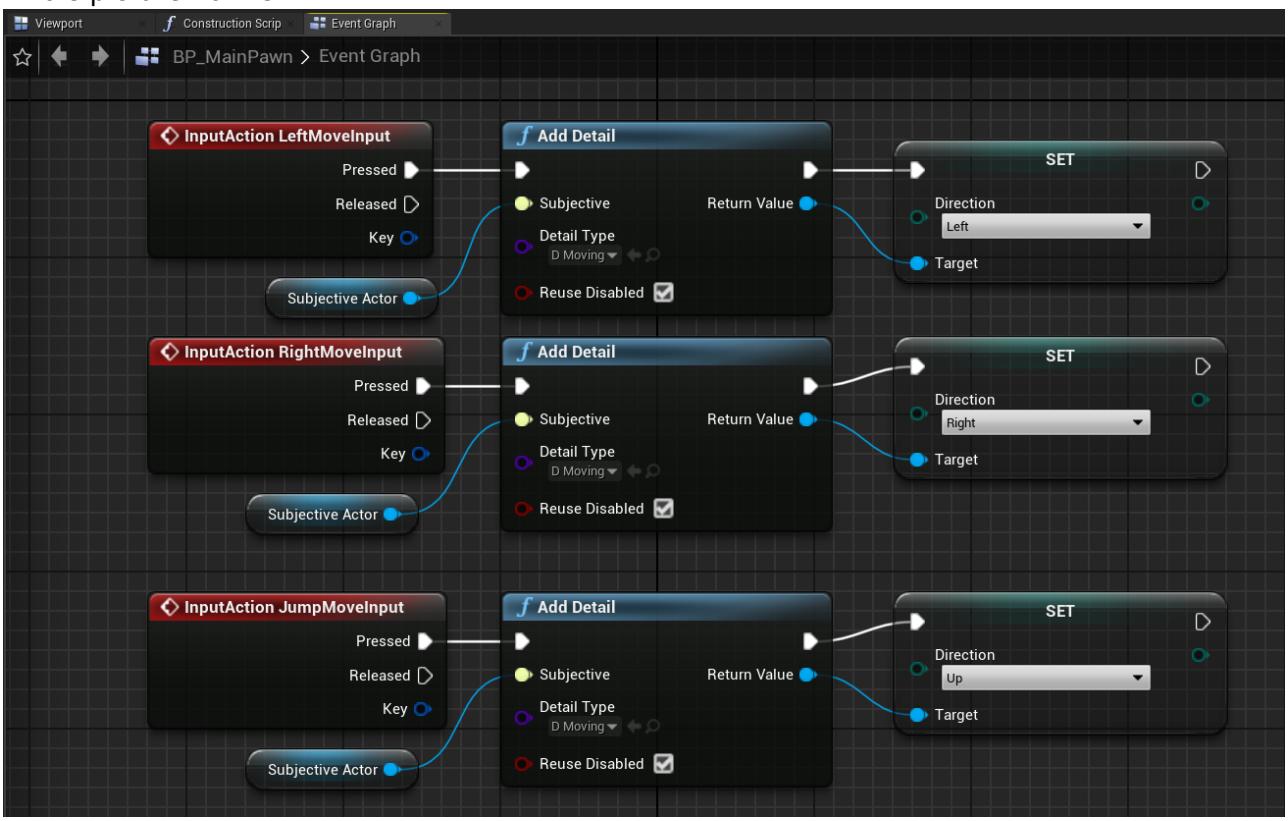

12. Now if you run the game, you will see that the camera actually working, but the cube don't move with pressing 'A', 'D' or 'W'. Let's fix it. Firstly, jump to 'BP_MainPawn' and in components list select 'SubjectiveActor' to see it's properties in 'Details panel'. In 'Details' property add new detail by clicking on the + button and selecting the detail class. Add two details like on the photo below:



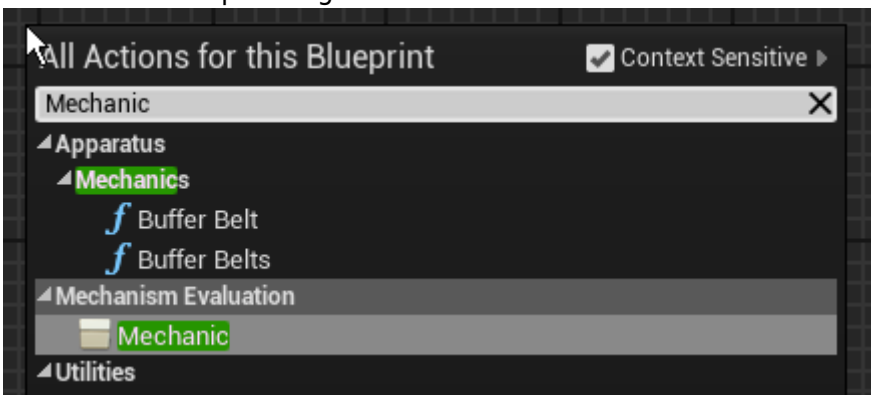

13. As you can see, you can easily add or remove details from 'BP_MainPawn' Actor component; no worth how they are ordered. You're also able to see the variables of the details and change their values. Note that if you change the value of the 'Speed' in the list, it won't be changed in

the ‘BP_Moveable’ detail, because here in the list the **real objects** of classes are represented while in the BP Editor of ‘BP_Moveable’ you can change only the default values of variables, which are the **options of class** itself. In the list of details **change Speed** to 500.

14. It’s better to do so in Controller but for brevity let’s do it here. Being in the BP Editor of the ‘BP_MainPawn’ navigate to ‘Event Graph’ and add our 3 input events we have declared previously (see step 6). Create the node ‘Get SubjectiveActor’, drag from it 3 new nodes, which are ‘Add detail’ and in the Detail Type field choose ‘D_Moving’. After that you can see the returning type of each node currently is ‘D Moving Object Reference’. I.e. after the detail was added you can ‘promote it to variable’ and call functions from it or access variables. In our case we will change the Direction variable to corresponding value. **Don’t forget to make checkboxes ‘Reuse inactive’ to true** (about why we should do so, we will talk later). The whole picture now is:



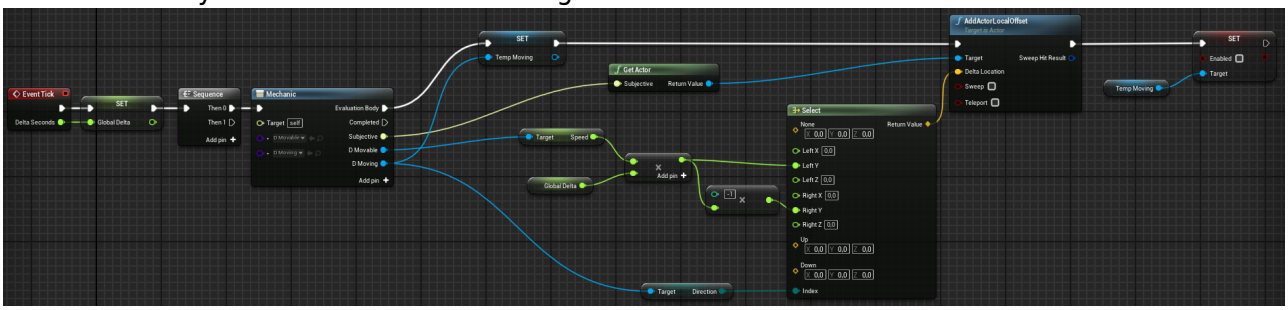

15. All we need to do now is to realize the game mechanics in the ‘GameMode’. So open BP Editor of ‘BP_Mechanism’ and in the Graph delete all nodes except of ‘Event Tick’. Promote the ‘Delta Seconds’ to global variable ‘GlobalDelta’. And now we need to iterate other subjects and for each of them check if it complies with special case or not. So, for that drag the next node ‘Sequence’ and from 1st its’ output drag the filament and find ‘Mechanic’ node.




16. As you can see, ‘Mechanic’ gets as input the ‘Mechanical Interface’, which is in fact our ‘BP_Mechanism’. This node will iterate over all entities with the specific set of details enabled/disabled. Currently now there is no such an entity, which will comply with the empty

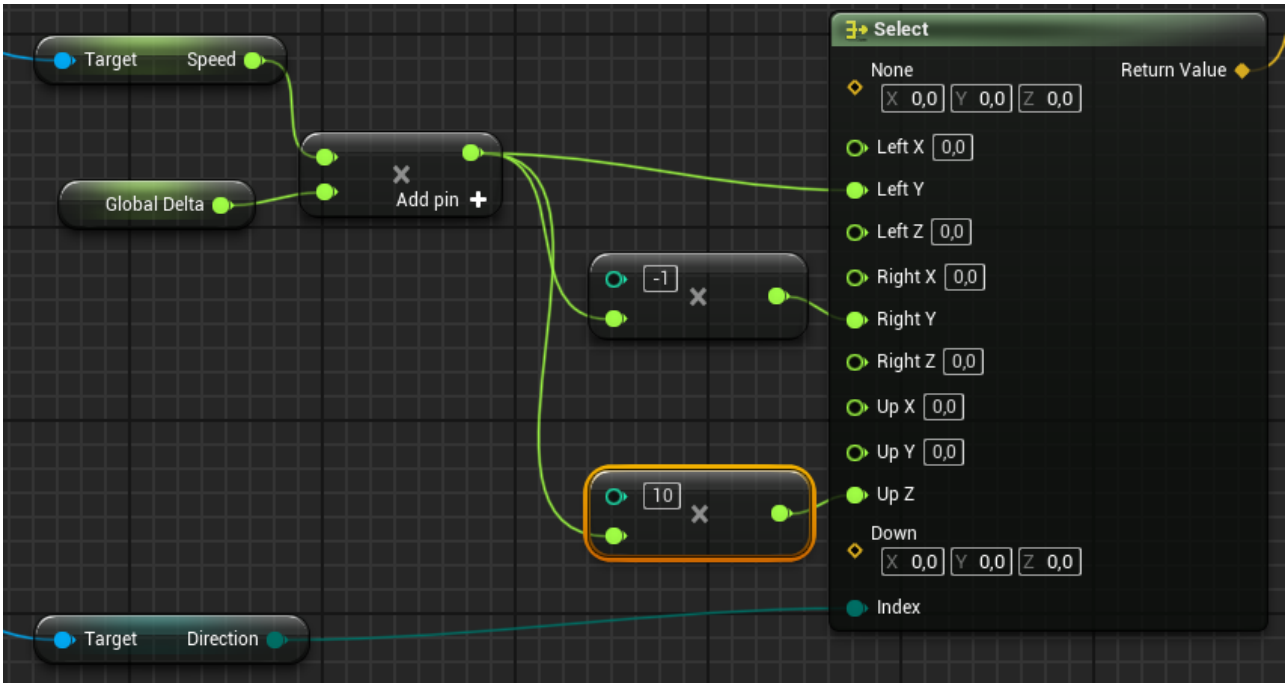
requirements. ‘Evaluation Body’ pin will be executed for each entity, but after all entities with the pointed set were processed, the ‘Completed’ pin will be executed. RMB click on the node and you see in it the last 2 items with these titles: ‘Add Detail pin’ & ‘Add non-Detail pin’. You can click these items several times and see that each time you do so, a new pin with a dot or exclamation mark to node. By using this technic and choosing detail type in ‘select class’ you declare the vector of 0es and 1es, you tell the mechanism, which subjects should be processed.


For example, add two dot-pins to the node and delete all the others (by clicking RMB and selecting ‘Remove a type/detail pin’). Choose their types as Moveable & Moving. Promote ‘Moving’ detail to variable ‘TempMoving’ - or you are to get something like macaroni. From ‘Subjective’ output pin drag ‘Get Actor’ pure function. It is also provided by Apparatus and returns the actor what is currently processed. From the function output drag ‘AddActorLocalOffset’. From ‘D_Moving’ pin drag ‘Direction’ variable and make ‘Select’ block to easily choose the corresponding vector depending on the direction given in the detail. ‘Left & ‘Right’ cases split into components and fill the Y-component of left case with the ‘Speed’ variable obtained from ‘Moveable’ detail and multiplicated with ‘GlobalDelta’ variable. For the right case use the same value but with the opposite sign. Beyond the ‘AddActorLocalOffset’ place the deactivation of ‘TempMoving’ detail be setting its’ variable ‘IsActivate’ to false. After all the actions you should reach something like that:



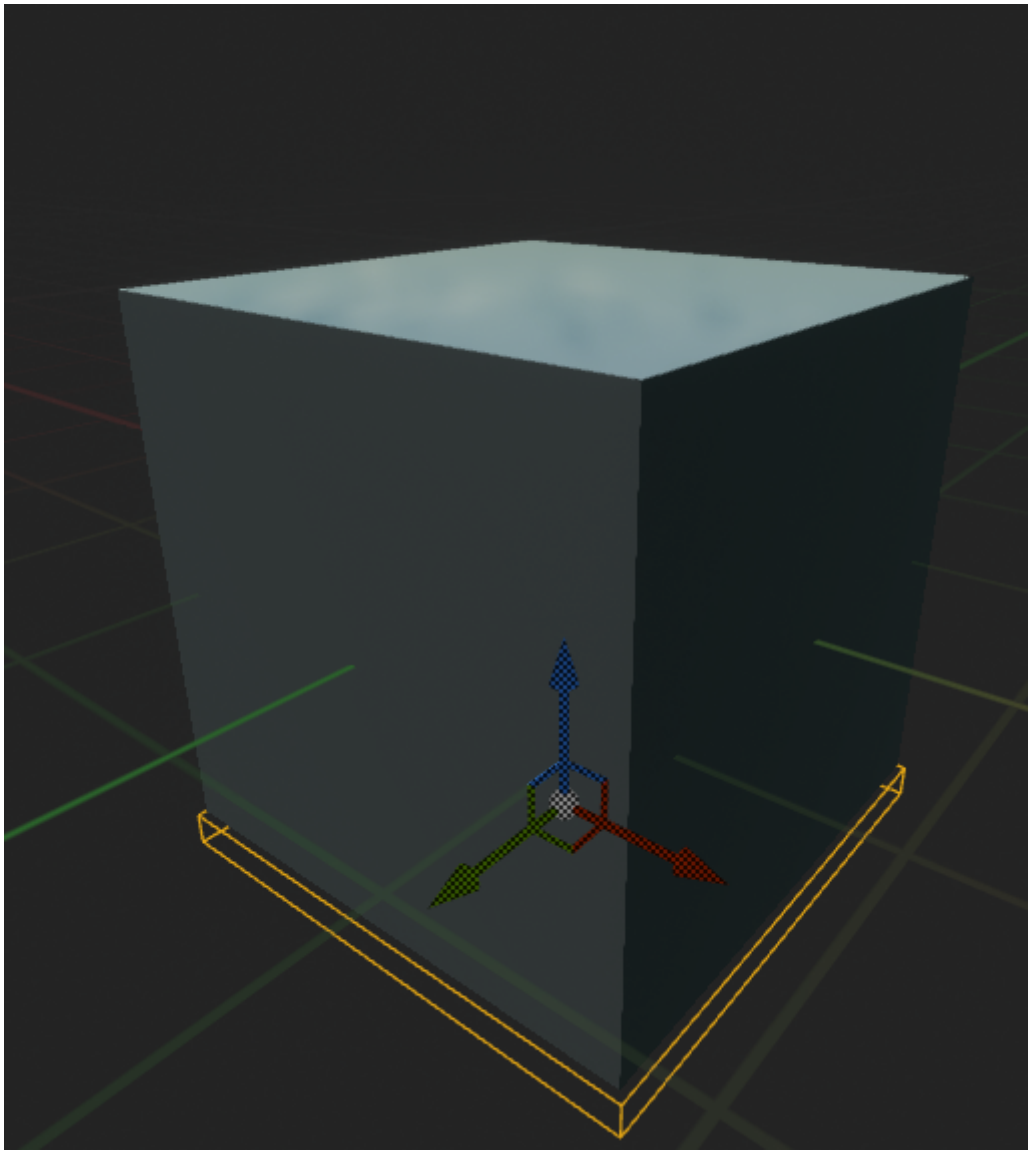
But what is going here? As you remember, we defined how we determine out keyboard input at the step 18. Here we just move each actor with the pointed details over its’ local Y-axis (for the camera view, it’s actually moving to the left if Y-axis is > 0 and moving to the right otherwise). So, depending on what the direction we obtained from the ‘Moving’ detail we move the actor across the scene. After doing so, we disable the Moving detail, so it won’t be moving to the side forever. Ok, but what will happen when the player will press ‘D’ or ‘A’ button the second time? Remember the checkbox we scored. ‘Reuse inactive’ in fact means that if the ‘Subjective’ has the disabled detail then the function will activate it and returning as its’ output. So now you can run the game and check if it works. Use ‘A’ & ‘D’ keys to move the box around.

17. Make a few changes: and you can jump.

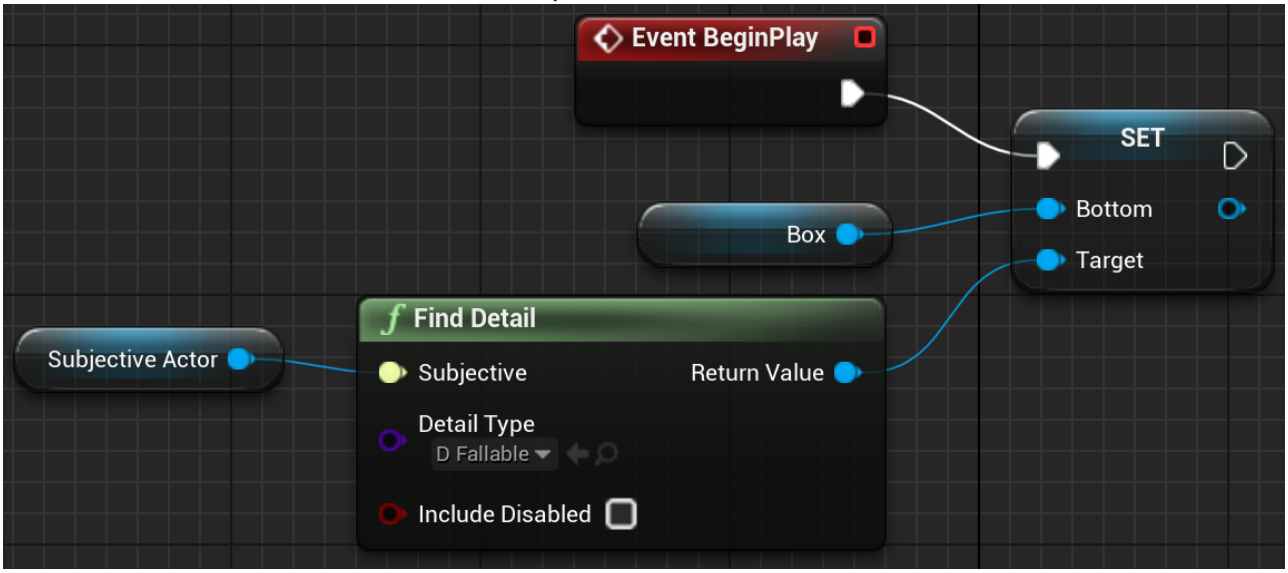



But the box isn't falling. Because we haven't declared necessary logic in Mechanism. So let's do that!

18. Move back to 'BP_MainPawn' and add to 'DefaultSceneRoot' new component 'BoxCollision'. Use next transform:
    - Location (X=0.000000,Y=0.000000,Z=-13.522278)
    - Scale (X=0.400000,Y=0.400000,Z=0.025000). After that in the 'Collision section' select 'OverlappAll' collision preset. But in the 'BP_MainPawn' editor the picture should be:



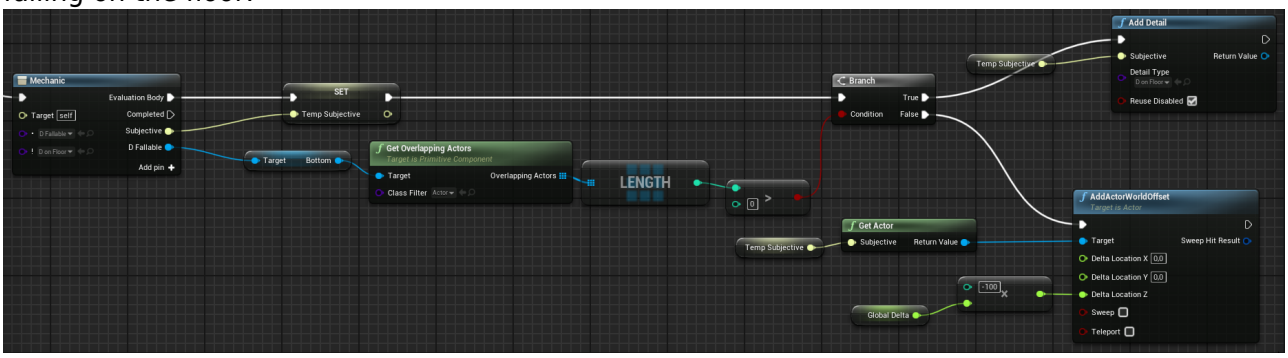
19. Now navigate to Graph and on ‘BeginPlay’ from ‘SubjectiveActor’ drag ‘find Detail’ function and set the bottom variable to the box of the pawn:




We are ensured that out Subjective will have that detail but you should understand the cases, when the output value of the function should be checked out.

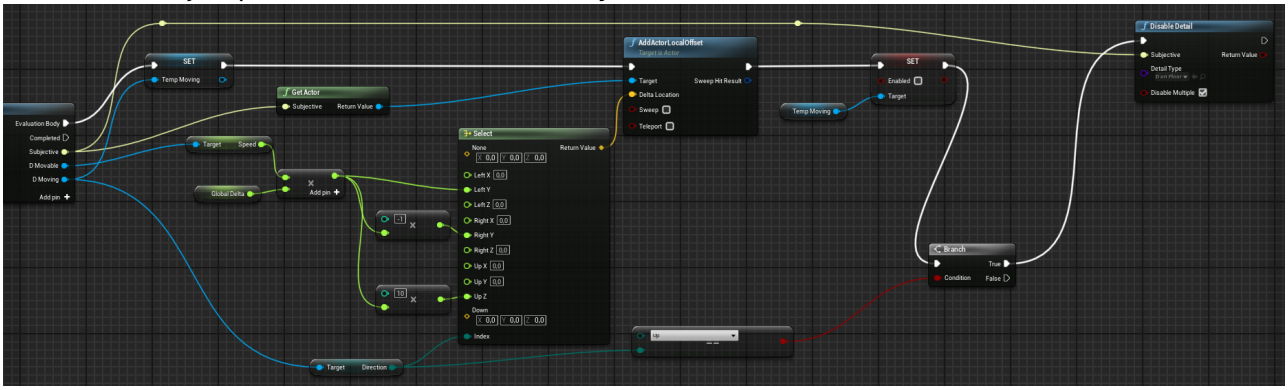
20. Navigate to the level, choose ‘Floor’ ‘StaticMeshActor’ and inside its details panel find the property ‘Generate Overlap Events’ and turn it on.
21. In the BP editor of ‘BP_Mechanism’ by dragging from sequence create next logic to provide

falling on the floor:




If the Subjective has enabled detail ‘Fallable’ & disabled detail ‘On Floor’ and if its’ bottom overlaps with any actor — then we add to it a detail ‘OnFloor’, else — move it down (like it’s falling).

22. And the last one — to begin the falling process we need disabling the detail ‘On Floor’ from the Pawn when it jumps. You can do it in that way:




If ‘Direction’==‘Up’ then we disable the detail by using Apparatus function. Now you can jump and fall.

## Conclusion

Apparatus is a really nifty plugin for our beloved UE, providing us with a bunch of new programming principles and techniques. You can use it in your game development production pipeline and extend its capabilities even furtherby declaring and implementing your own C++ classes, adhering to the necessary Apparatus interfaces.

The whole functionality of the plugin can’t be demonstrated on a little tutorial like this, but the main purpose of this article is exactly to introduce the beginners to the ECS approach and some of the main features of Apparatus!

## Links

- The resulting project on GitHub
- A more complex sample on GitHub
- Online API Reference