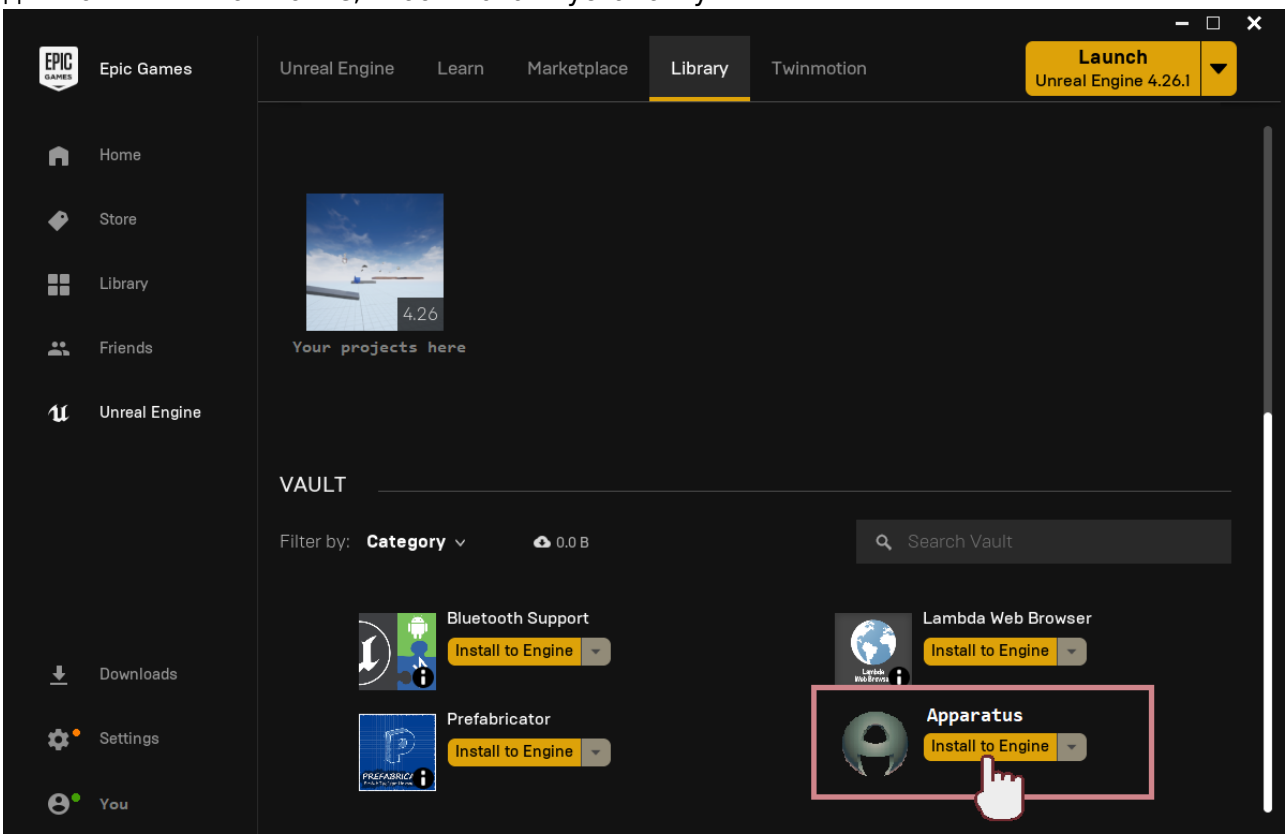
Apparatus: Введение

В этом непродолжительном уроке мы поговорим о использовании Apparatus-плагина в Unreal Engine. Вы создадите свою первую деталь и научитесь реализовывать игровую логику в специально отведённом Blueprint-классе. Здесь продемонстрированы самые главные особенности работы с плагином на примере простого двумерного платформера.

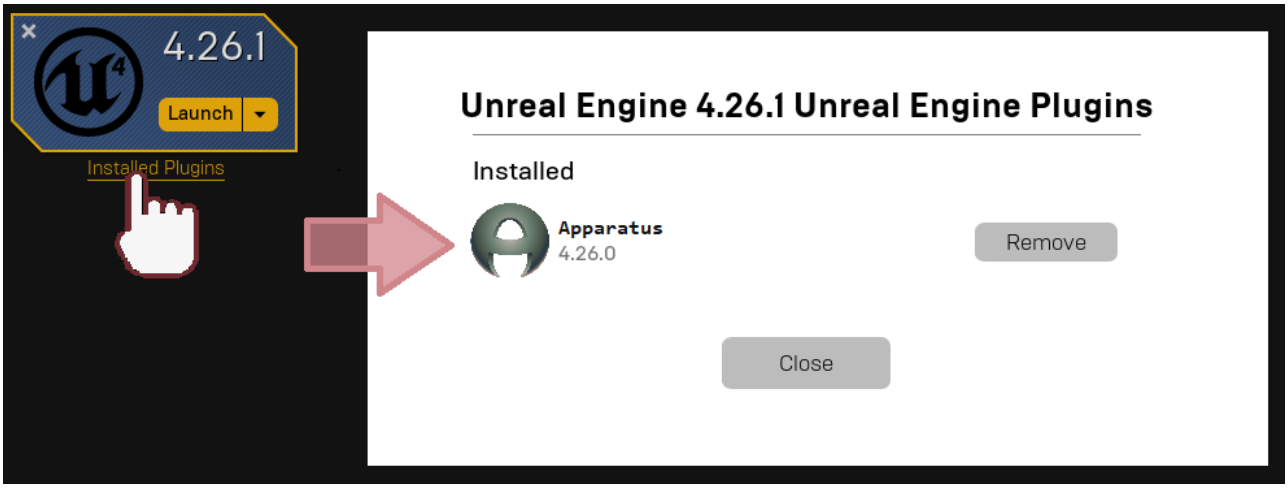
Для дальнейшего чтения необходимо понимать базовые концепции ECS-подхода. Для этого есть наша краткая справочка по ECS.

Установка плагина и активация

- Перед тем, как создать новый проект, Вам потребуется добавить плагин к игровому движку. Чтобы это сделать, пожалуйста, загрузите Epic Game Launcher, в левом меню выберете Unreal Engine, затем в верхнем меню - Библиотека. Промотайте вниз и под секцией ‘Хранилище’ найдите плагин Apparatus с жёлтой кнопкой ‘Установить на движок’. Клик по кнопке, чтобы начать установку.

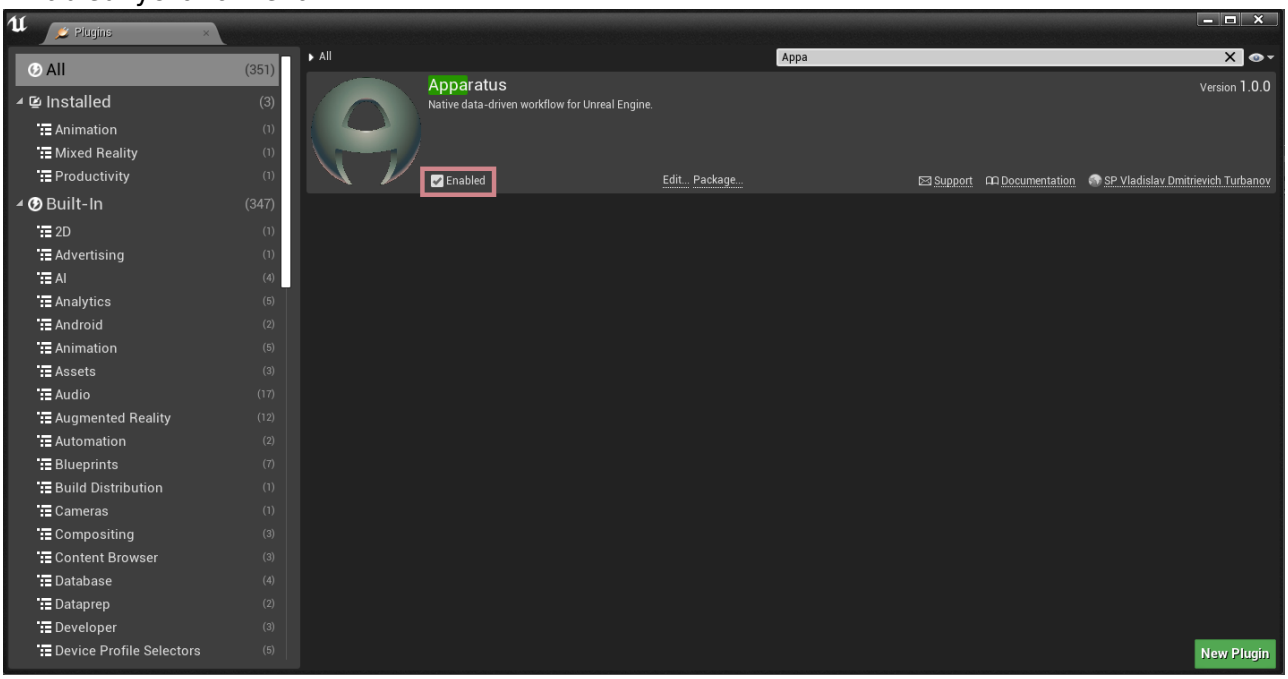


- В открывшемся окне необходимо выбрать версию Unreal Engine. Обращаем внимание, что, на настоящий момент, официально поддерживаемые версии - 4.26.1 и выше. После клика по ‘Установить’ подождите пару минут, пока загрузчик встроить код плагина в движок. Когда установка завершилась, можно проверить её успешность кликом по ‘Установленные дополнения’ под версией движка.



Создание проекта

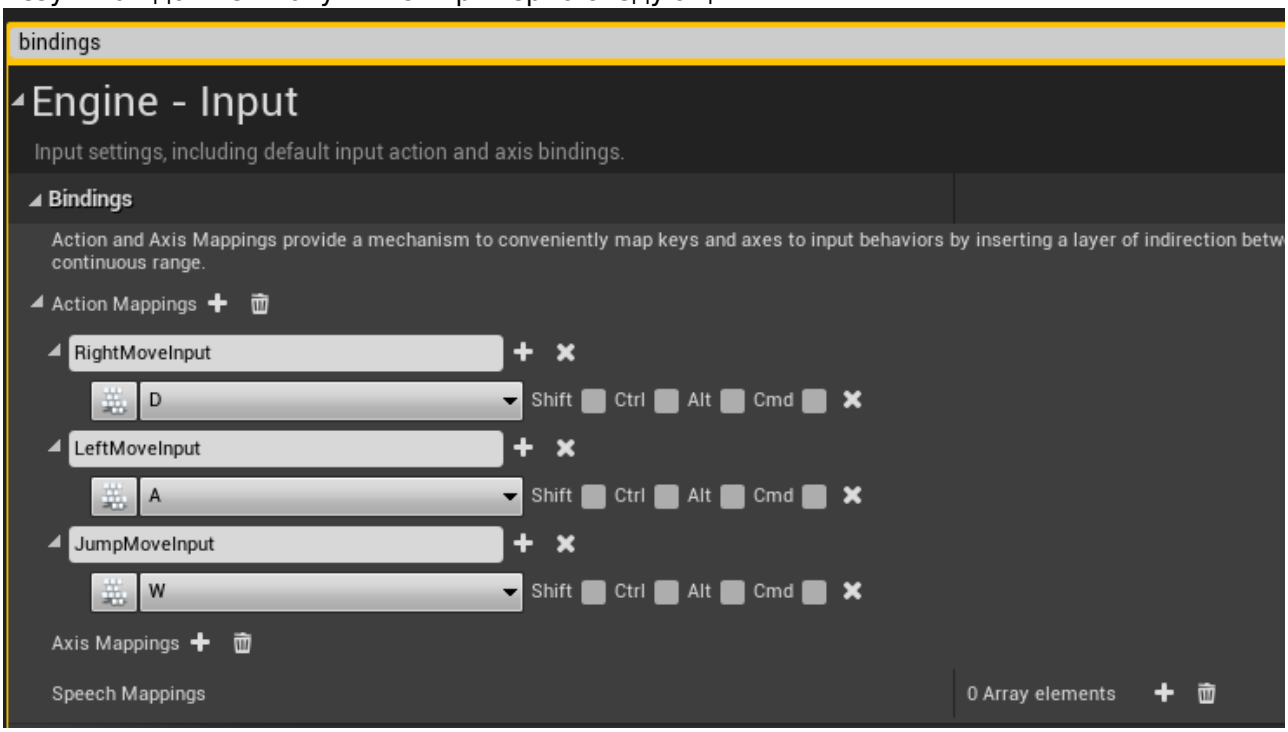
- Теперь надо создать новый проект. Выберете пустой шаблон, так как в этом уроке мы всё настроим с нуля. Остальные опции можно оставить без изменений. Мы назовём модельную игру “ApparatusLearn”, ну а Вы можете использовать любое имя, какое нравится.
- Когда проект создался и открылся, можете проверить, выбран ли плагин в настройках. Для этого в верхнем меню выберете ‘Edit’ → ‘Plugins’. Затем напечатайте ‘Apparatus’ в строке поиска (или промотайте вниз до ‘Workflow’ секции). Убедитесь, что галочка ‘Enabled’ установлена:



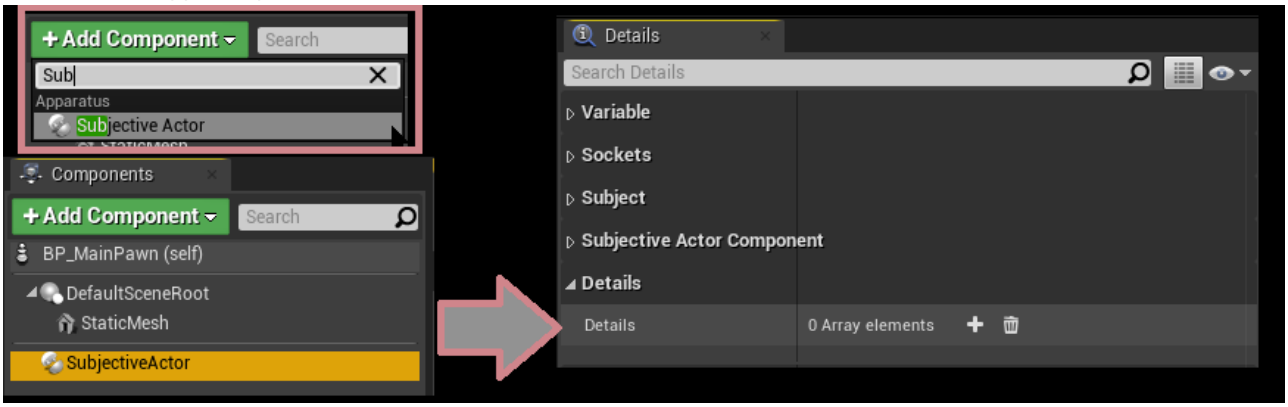
Начало работы с плагином

- Хорошо, перво-наперво нам предстоит добавить привязку клавиш, чтобы понимать, когда надо добавлять необходимые детали к Actor’у. Чтобы это сделать, перейдём в ‘Edit’→‘Project Settings’ и печатаем ‘bindings’. Найдём секцию ‘Action Mappings’ и в неё добавим следующие клавиши:
  - ‘RightMoveInput’ – D на клавиатуре;

    - ‘LeftMoveInput’ – A на клавиатуре;
    - ‘JumpMoveInput’ – W на клавиатуре.
- Результат должен получиться примерно следующим:



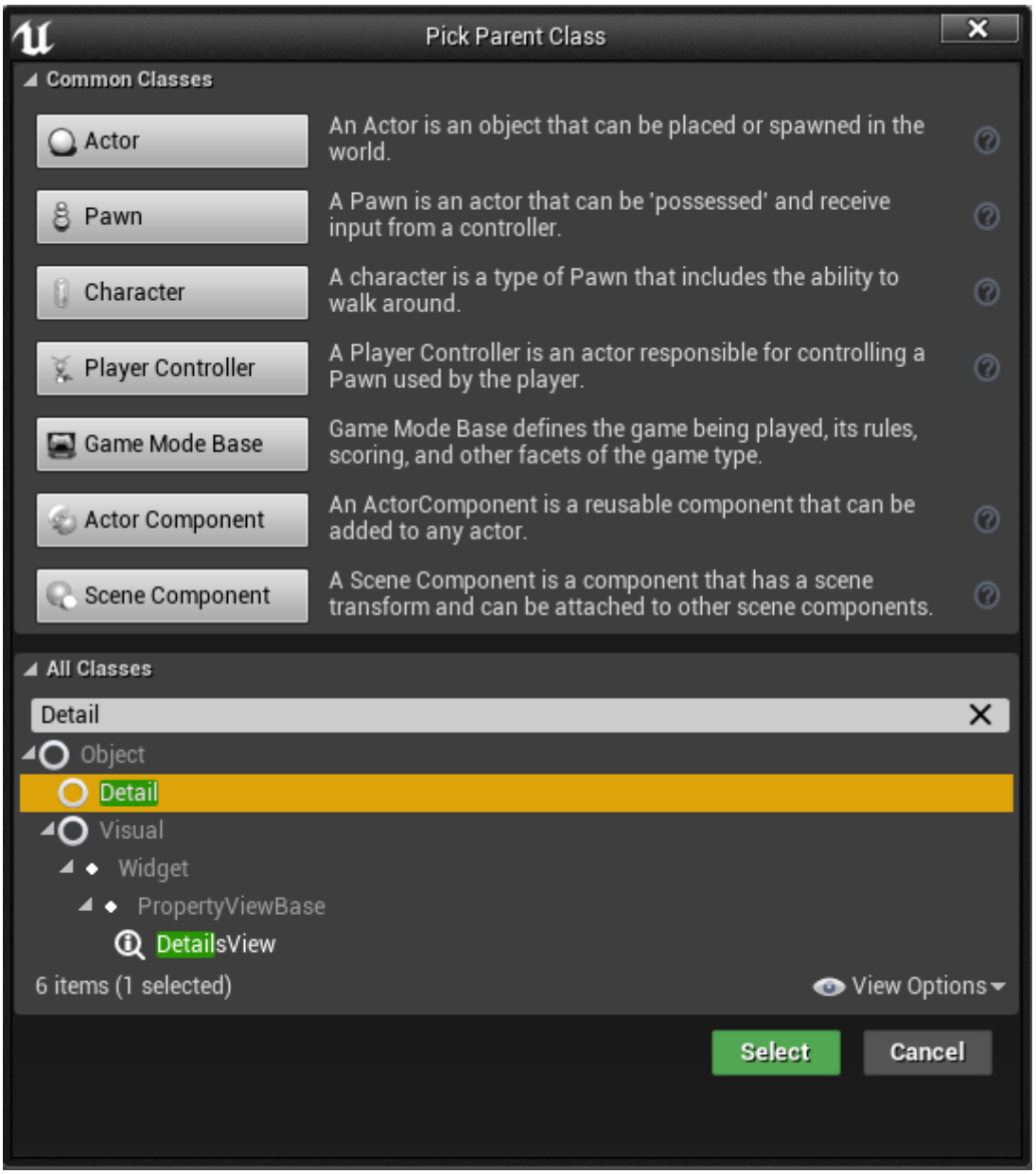
- Мы начнём с создания новой ‘пешки’ (Pawn Blueprint). Для этого нажмём зелёную кнопочку ‘Add/Import’ в Content-браузере. Выберем ‘Blueprint Class’→‘Pawn’ и дадим ему имя ‘BP_MainPawn’ (подробнее про именование ассетов можно посмотреть в стиль-руководстве). После создания нового блупринт-класса, прожмите Ctrl+S, чтобы сохранить только что созданный объект. Теперь двойным нажатием по иконке откроем редактор блупринтов (если Вы в первый раз сталкиваетесь с Content-браузером, советуем проверить официальную документацию движка). Переходим в Event-граф и удаляем все ноды через Ctrl+A и Del. В дальнейшем подразумевается, что Вы знакомы и с редактором Blueprint'ов; далее - BP).
- В редакторе BP перейдём в Viewport и к списку компонентов класса добавим ‘StaticMesh’. В панели деталей справа для ‘Static Mesh’-свойства выбираем ‘Cube’ mesh-ассет. Чтобы пешка выглядела более приятной, для ‘Element 0’ свойства выбираем материал ‘BrushedMetal’. А сейчас добавьте ‘Subjective Actor’ компонент (предоставляемый Apparatus’ом) к нашей пешке и посмотрите в панель деталей, чтобы ближе познакомиться с новым Actor-компонентом. В этом уроке нам потребуется только свойства под секцией ‘Details’:



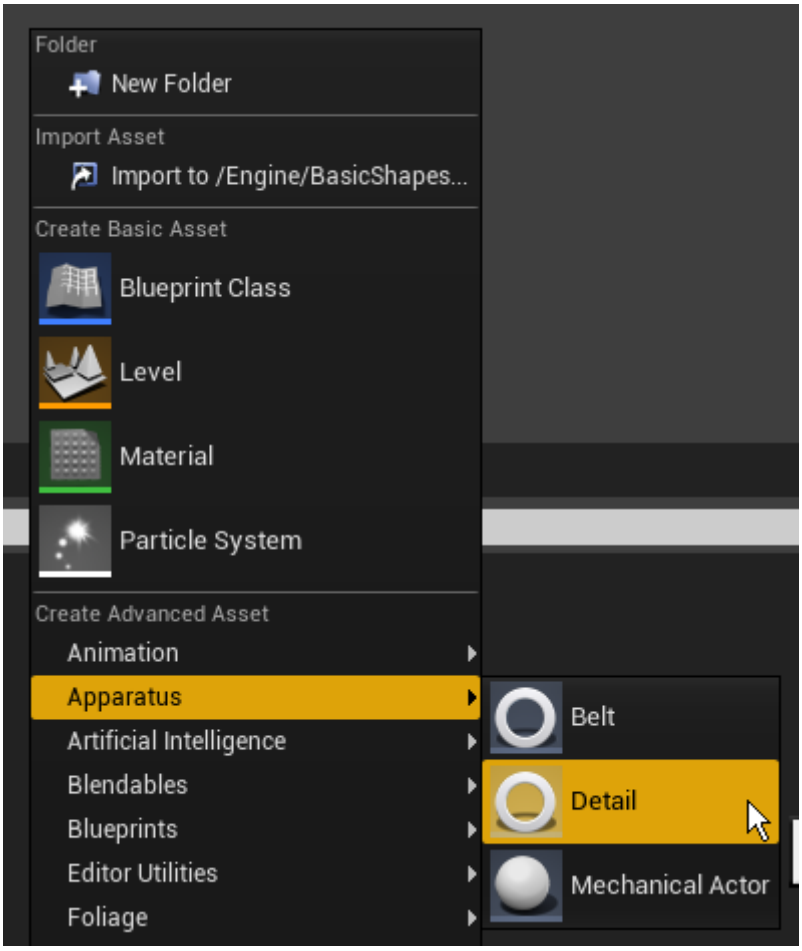
Как Вы уже догадались, детали мы будет помещать и настраивать именно здесь.

- Ctrl+Shift+S чтобы сохранить всё, и скомпилируем BP. Вновь откроем Content-браузер и создадим новый BP, но на этот раз раскроем панель ‘All classes’ и найдём там класс

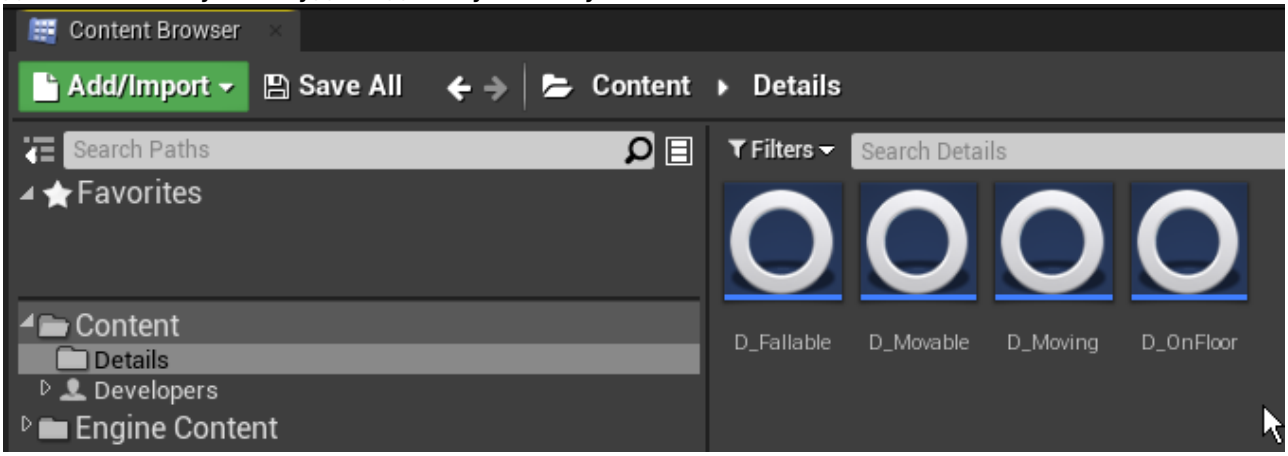
‘Detail’:



Вы можете также создавать детали через ‘Create Advanced Asset’ секцию в меню Content-браузера (правой кнопкой мыши по пустому месту в папке):



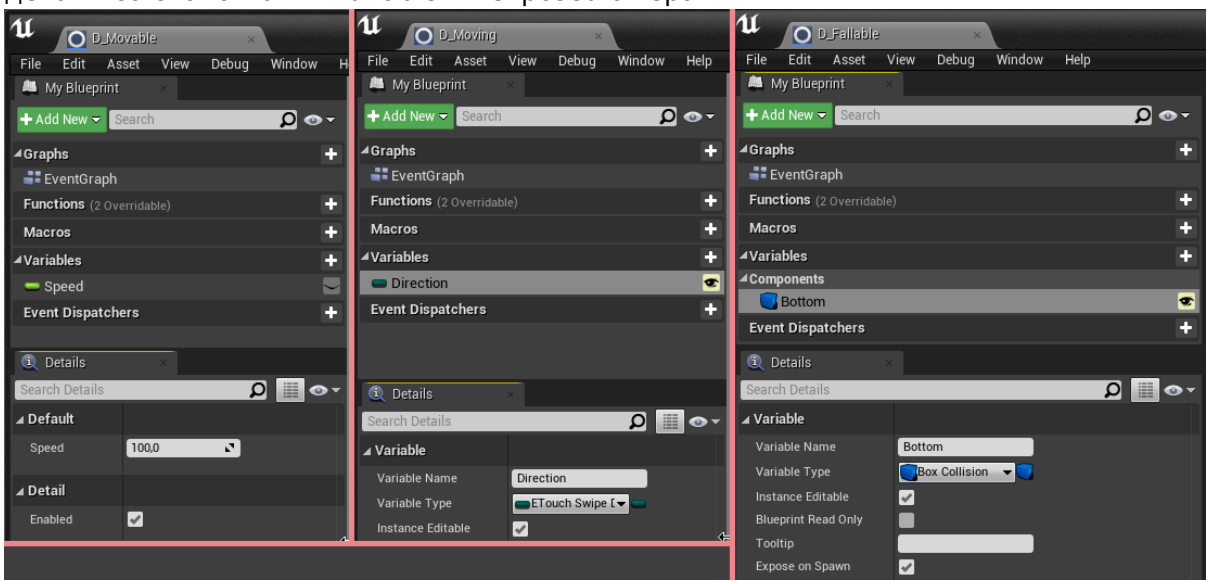
- Вообще говоря, Вы можете создать сколько угодно деталей, но для нужд этого туториала пригодятся следующие:
  - `D_Moveable`,
  - `D_Moving`,
  - `D_OnFloor`,
  - `D_Fallable`.
- Чтобы поддержать организованность проекта, переместите все созданные классы деталей в какую-нибудь отдельную папку ‘Details’.



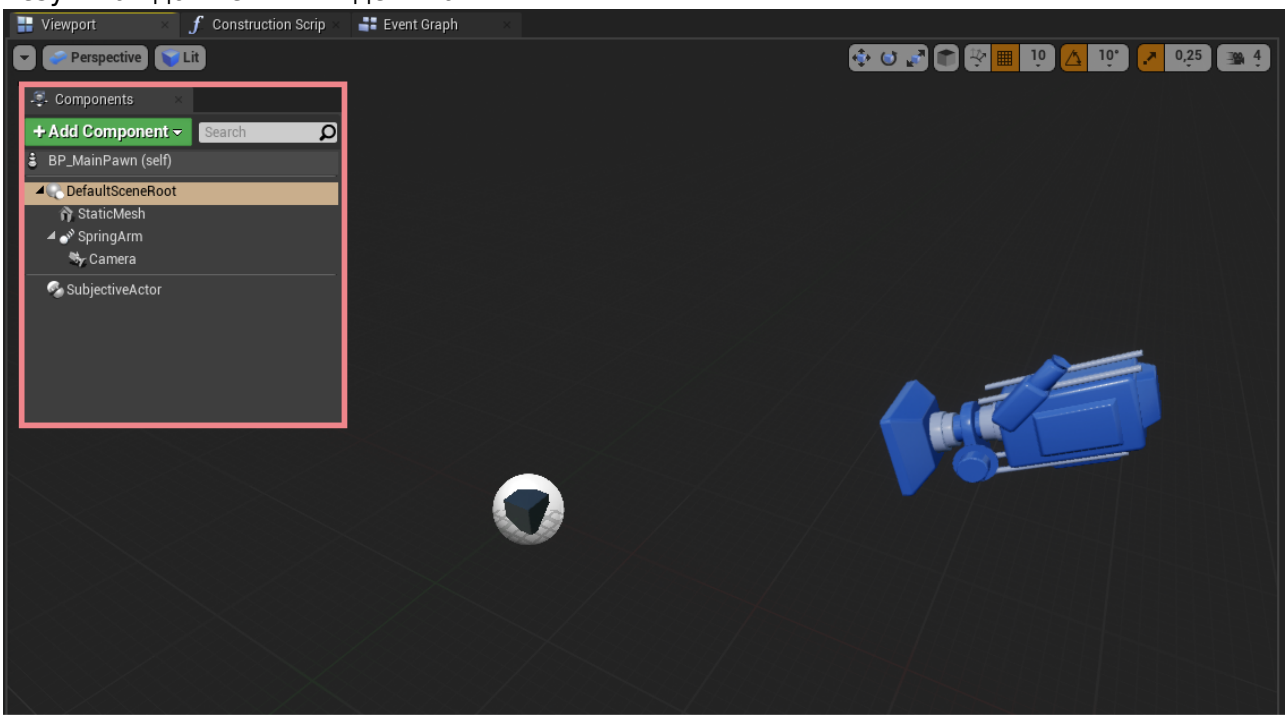
Откройте любую деталь в BP-редакторе. Несложно видеть, детали в Apparatus - это на самом деле обычные BP-классы, где можно по собственному усмотрению объявлять переменные, заводить макросы и функции. Так же плагин предоставляет 2 перегружаемые функции (override events): ‘Activated’ и ‘Deactivated’, которые вызываются, когда устанавливается соответственное значение флага ‘Enabled’. В следующих деталях добавьте необходимые переменные:

- Float ‘Speed’ в `D_Moveable` с значением по умолчанию `100.0`.

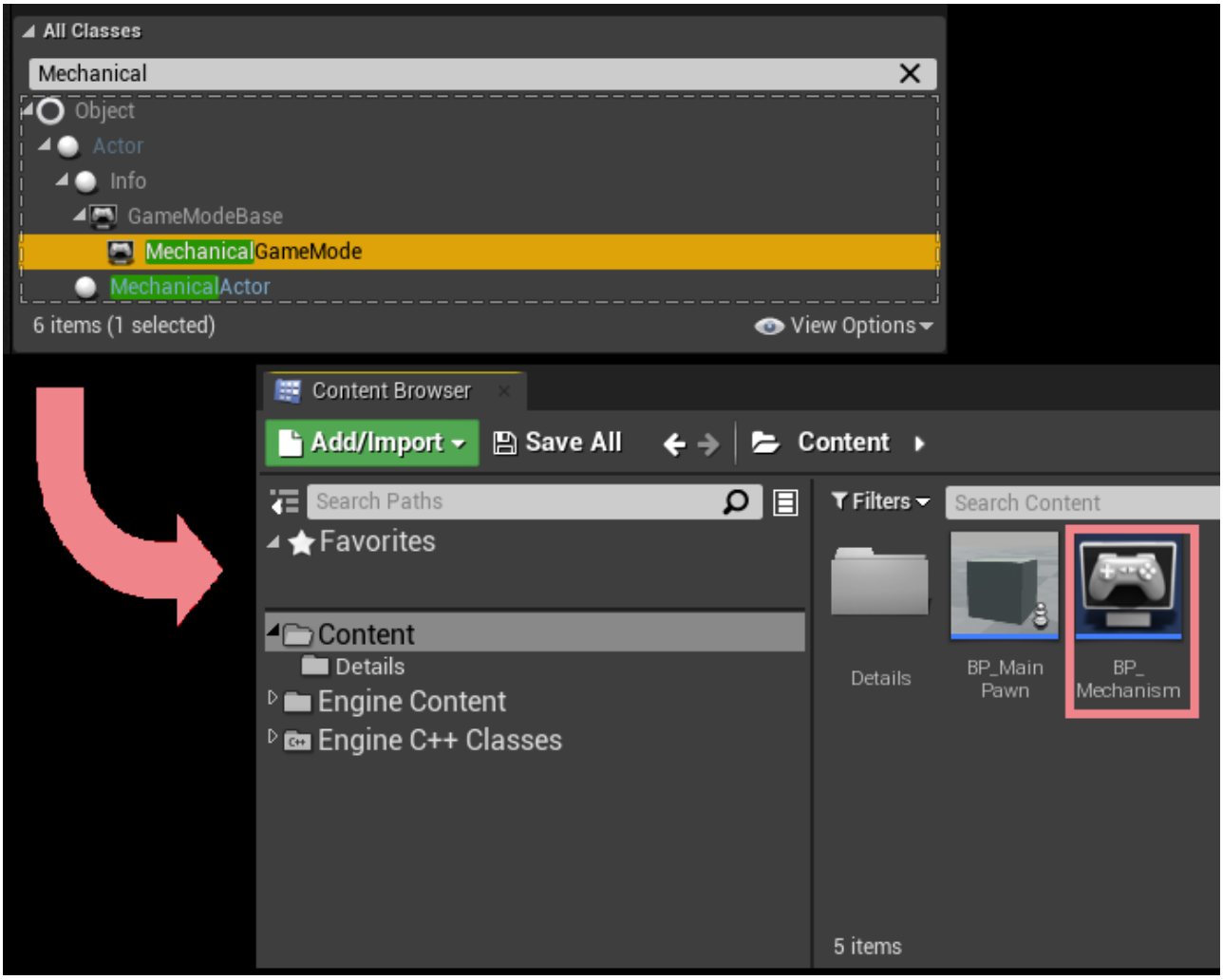
- Перечисление `ETouch Swipe Direction` с именем `Direction` со свойством ‘Editable’ в `D_Moving`.
- Ссылку типа `Box Collision Object Reference` с именем ‘Bottom’ к `D_Fallable` детали со свойствами ‘Editable’ и ‘exposed on spawn’.



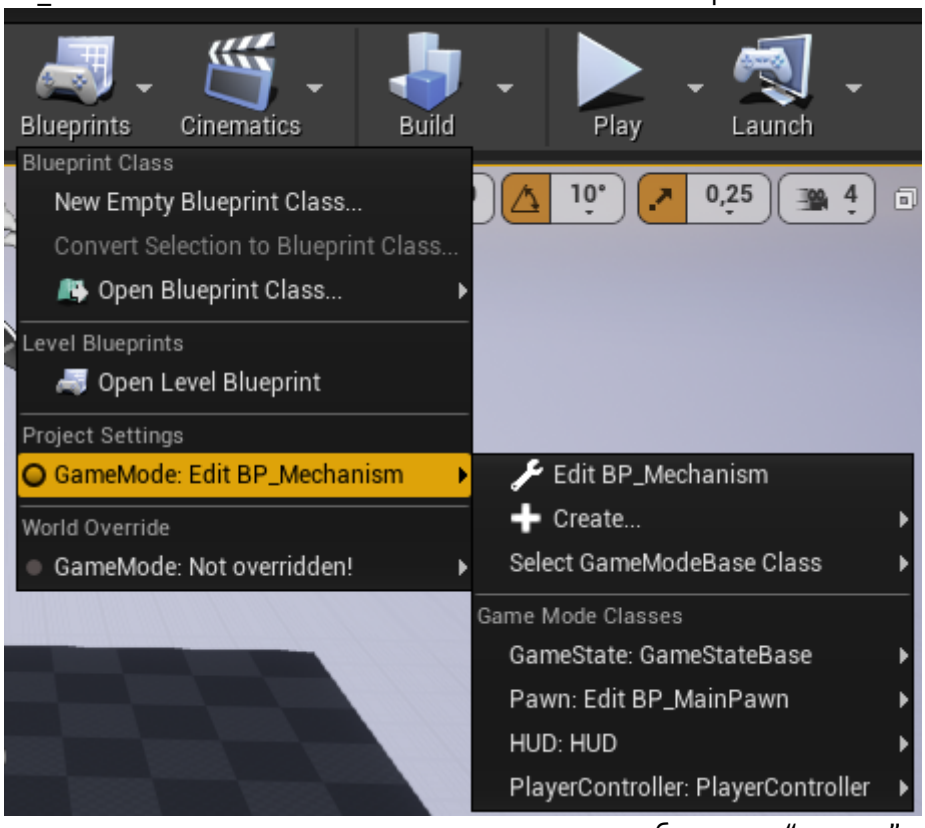
- Перейдя обратно к классу ‘BP_MainPawn’, сделайте куб немного поменьше (например, скопируйте эту строчку вместе с скобками: `(0.25, 0.25, 0.25)` - и, щёлкнув правой кнопкой мыши по вектору scale, вставьте скопированное значение значение; если всё сработает правильно, значения компонент установятся в надлежащие числа). Теперь добавим Actor-component ‘Spring Arm’ и привяжем его к ‘DefaultSceneRoot’, затем добавим также камеру, привязав её к ‘Arm’ (убедитесь, что при этом вектора масштаба на новых компонентах остались в значениях по умолчанию `(1,1,1)`). Немного повернём ‘Arm’ по Z-оси на 180°, а после и по Y-оси, но уже на -30°.
- Результат должен выглядеть так:



- Создадим новый ‘GameMode’ (или ‘GameModeBase’, - и тот и другой вариант приемлем) наследовавшись от ‘MechanicalGameMode’ (соответственно - ‘MechanicalGameModeBase’), новый класс назовём ‘BP_Mechanism’. Примерно так:

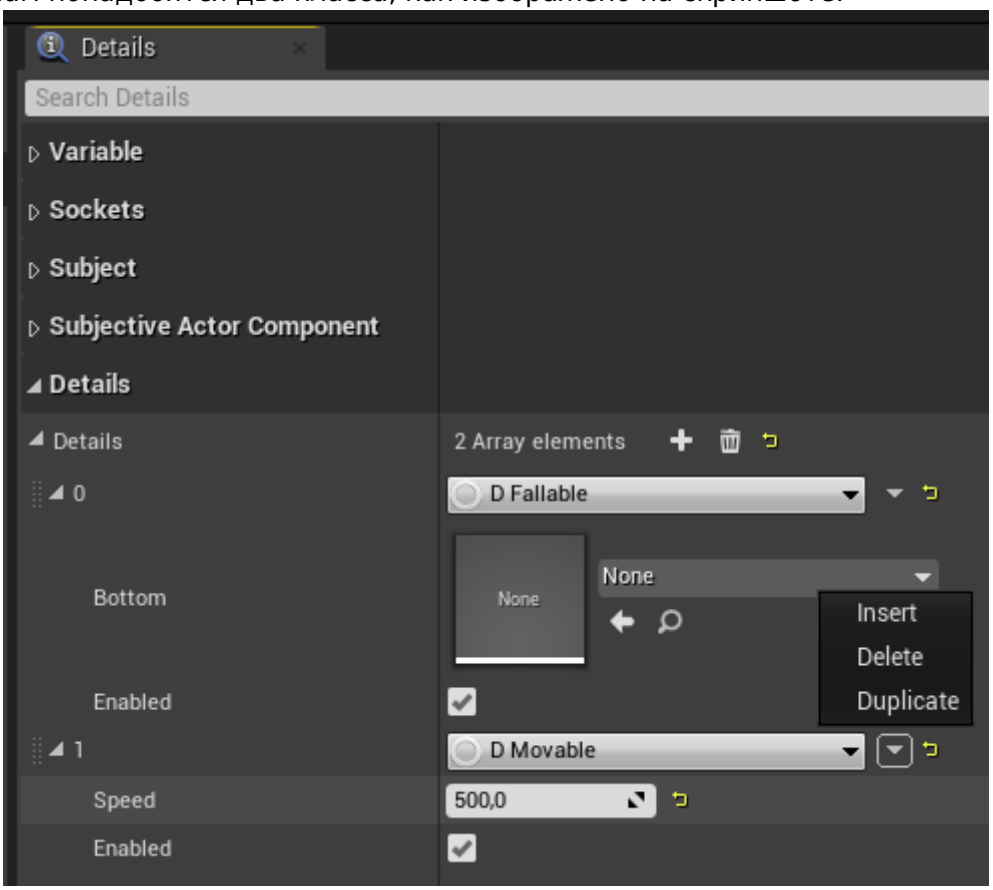


- Открываем ‘BP_Mechanism’ в редакторе и в панели деталей установим ‘Default pawn class’ в ‘BP_MainPawn’. Далее идём в настройки уровня: ‘Blueprints’→‘Project Settings : GameMode’ и выбираем ‘BP_Mechanism’ в качестве главного GameMode’а проекта:

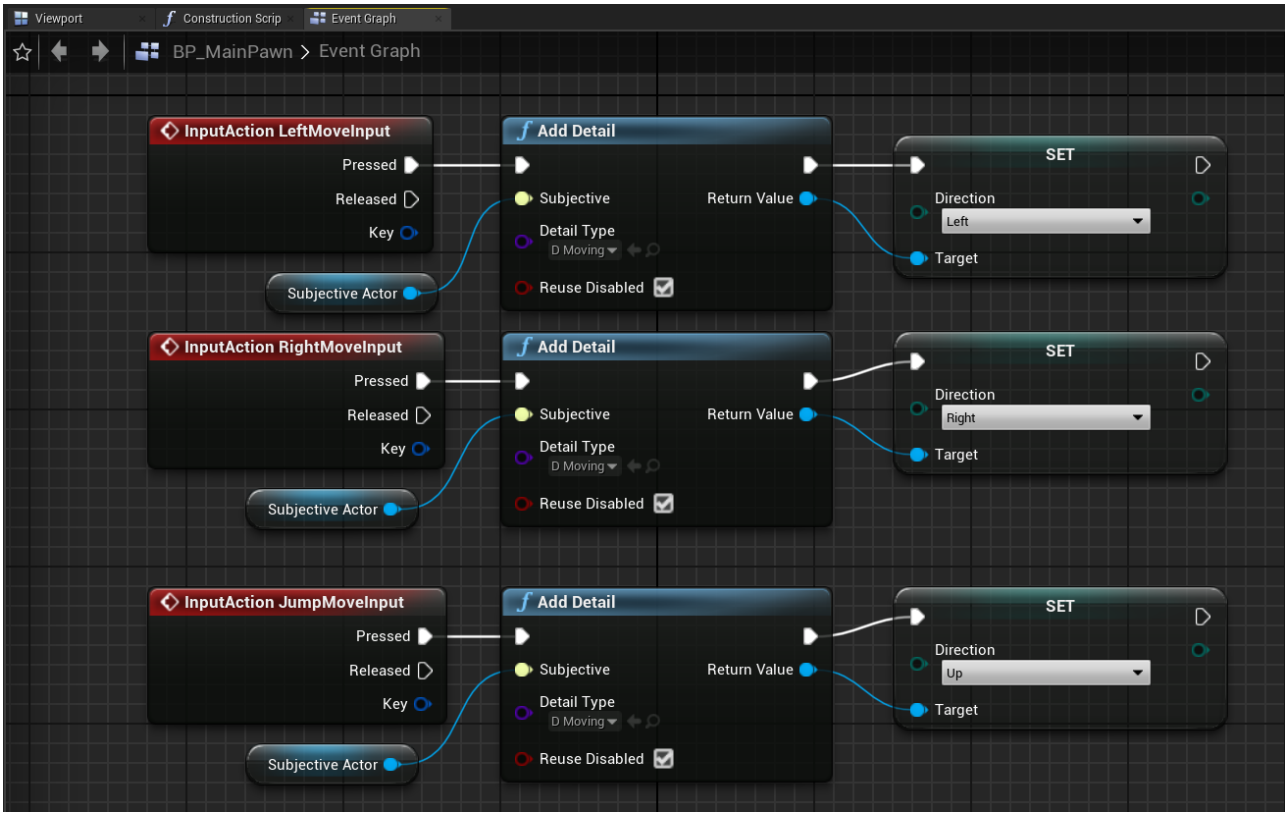


- Теперь, если запустить игру, можно видеть, что камера работает и “пешка” спавнится;

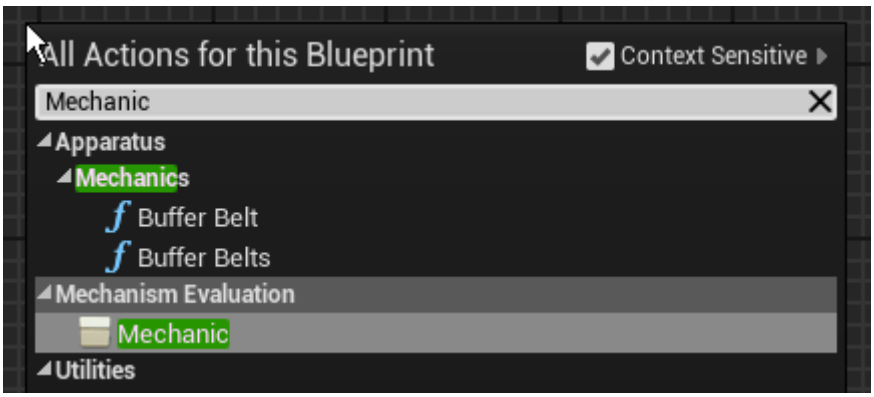
но куб не двигается по нажатию на A, D, W. Исправим это. Для начала, находим ‘BP_MainPawn’ в редакторе BP и в списке Actor-компонентов выбираем ‘SubjectiveActor’, чтобы его свойства отобразились на панели деталей справа. В этой панели находим свойство ‘Details’ и к нему при помощи кнопочки + добавляем новые детали и выбираем их типы. Нам понадобятся два класса, как изображено на скриншоте:



- Теперь Вы видите, что добавлять и удалять детали можно очень просто через настройки Actor’ компонента, причём совершенно не важно, в каком порядке они добавлены, в каком порядке в списке отображаются. Вы также можете видеть открытые переменные деталей и менять их значения по умолчанию. Заметим, что если, например, изменить параметр ‘Speed’ в списке, его default-значение не будет изменено в BP-редакторе ‘BP_Moveable’, потому как в списке представлены именно инстанциированные объекты класса детали. Изменим здесь значение Speed на 500.
- Правильней было бы сделать это в контроллере, но для краткости сделаем это здесь. Будучи в BP-редакторе пешки, перейдите в Event-граф и добавьте 3 события по нажатию клавиши, что мы настроили ранее (см. шаг 2). Создадим ноду ‘Get SubjectiveActor’ и вытащим из неё 3 другие: в окне поиска функций, пожалуйста, найдите ‘Add Detail’ и в качестве типа детали выберете ‘D_Moving’. После этого вы можете видеть, что выходной тип функции сменился на `D Moving Object Reference`. Иначе говоря, после того, как деталь была добавлена к сущности, можно преобразовать её в переменную (‘promote to variable’) и использовать в своём коде для вызова её функций или доступа к переменным. В нашем случае, мы обратимся к полю направления (‘Direction’) и установим соответствующие значения. **Не забудьте** установить флаги ‘Reuse Disabled’ (о том, зачем они, - чуть позже). Полная картина должна быть примерно такой:



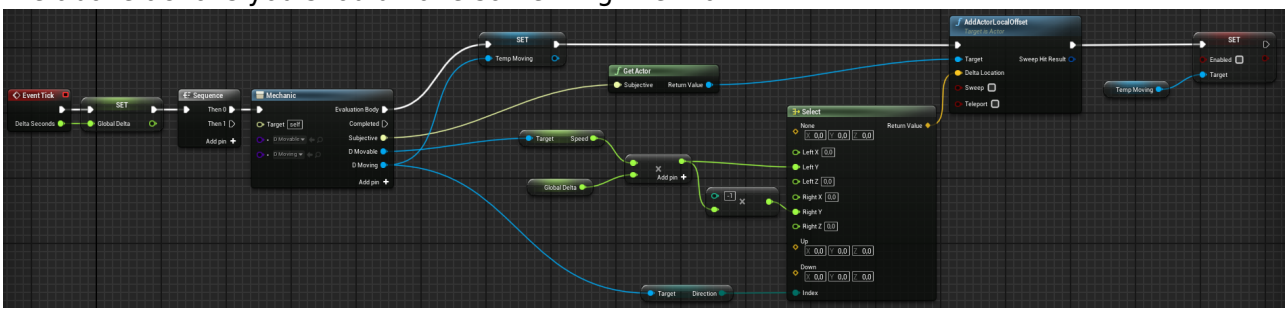
- Всё, что нам осталось сделать, - это реализовать игровые механики в нашем ‘GameMode’е. Итак, откроем редактор блупринтов ‘BP_Mechanism’а и в графе нод удалим все события, кроме ‘Event Tick’. Для удобства преобразуем `Delta Seconds` в глобальную переменную `GlobalDelta`. Теперь нам необходимо интегрироваться по всем сущностям (Subject’ам) и для каждого проверять, какой набор деталей на нём есть. Для этого, пожалуйста, вытяните следующую ноду ‘Sequence’ и из её первого output-pin’а вытянем ноду ‘Mechanic’.



- Как нетрудно видеть, эта нода получает на вход ‘Mechanical Interface’…
- As you see, ‘Mechanic’ gets a ‘Mechanical Interface’ as its input ‘self’, which is in fact our ‘BP_Mechanism’ is. This node will iterate over all of the entities (subjects) with a specific set of details enabled/disabled. As how it is from the starts, essentially all of the subjects which will comply with the empty requirements. ‘Evaluation Body’ pin gets executed for each complying subject. After all of complying subjects were processed, the ‘Completed’ pin gets executed. RMB-click on the node and in the opened context menu you will see two last items with these titles: ‘Add Detail Pin’ and ‘Add Excluded 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 is added to the node. By using this workflow and selecting the detail types in the added inputs you declare an evaluation filter, for the subjects to be processed in the mechanic.

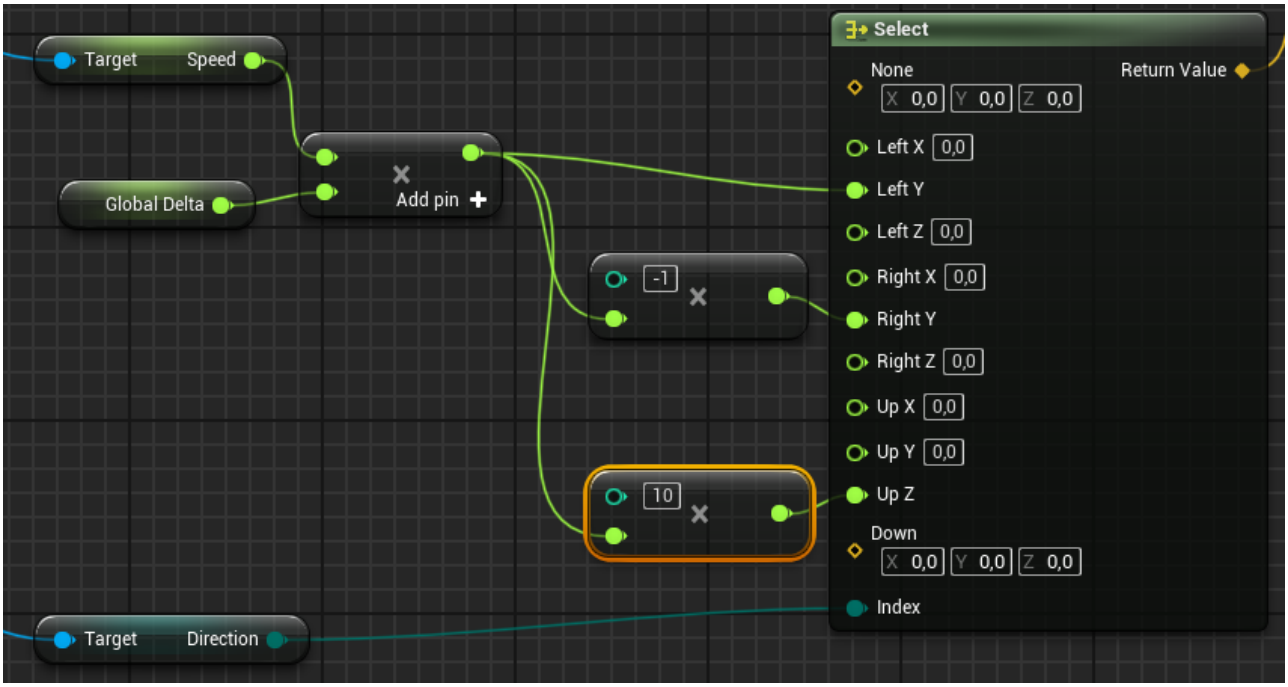


For example, add two including “dot”-pins to the node and delete all of the others (by RMB-clicking on them and selecting the ‘Remove Detail Pin’ option). Choose their types as Moveable and Moving. Promote the ‘Moving’ detail to a ‘TempMoving’ variable - or you’ll get yourself too much “noodles”. From the ‘Subjective’ output pin drag a ‘Get Actor’ pure function. That is also provided by Apparatus and returns the actor that is currently being processed. From this function’s output drag a ‘AddActorLocalOffset’ node. From the ‘D_Moving’ pin drag the ‘Direction’ variable and make ‘Select’ block to easily choose the corresponding vector depending on the direction currently active in the detail. ‘Left and ‘Right’ cases are split into components and fill the Y-component of the left-side case with the ‘Speed’ variable obtained from the ‘Moveable’ detail and multiplicated by the ‘GlobalDelta’ variable. For the right-side case use the same value just with an opposite sign. Below your ‘AddActorLocalOffset’ node place the deactivation of the ‘TempMoving’ detail be setting its state ‘Enabled’ to false. After all the above actions you should have something like that:



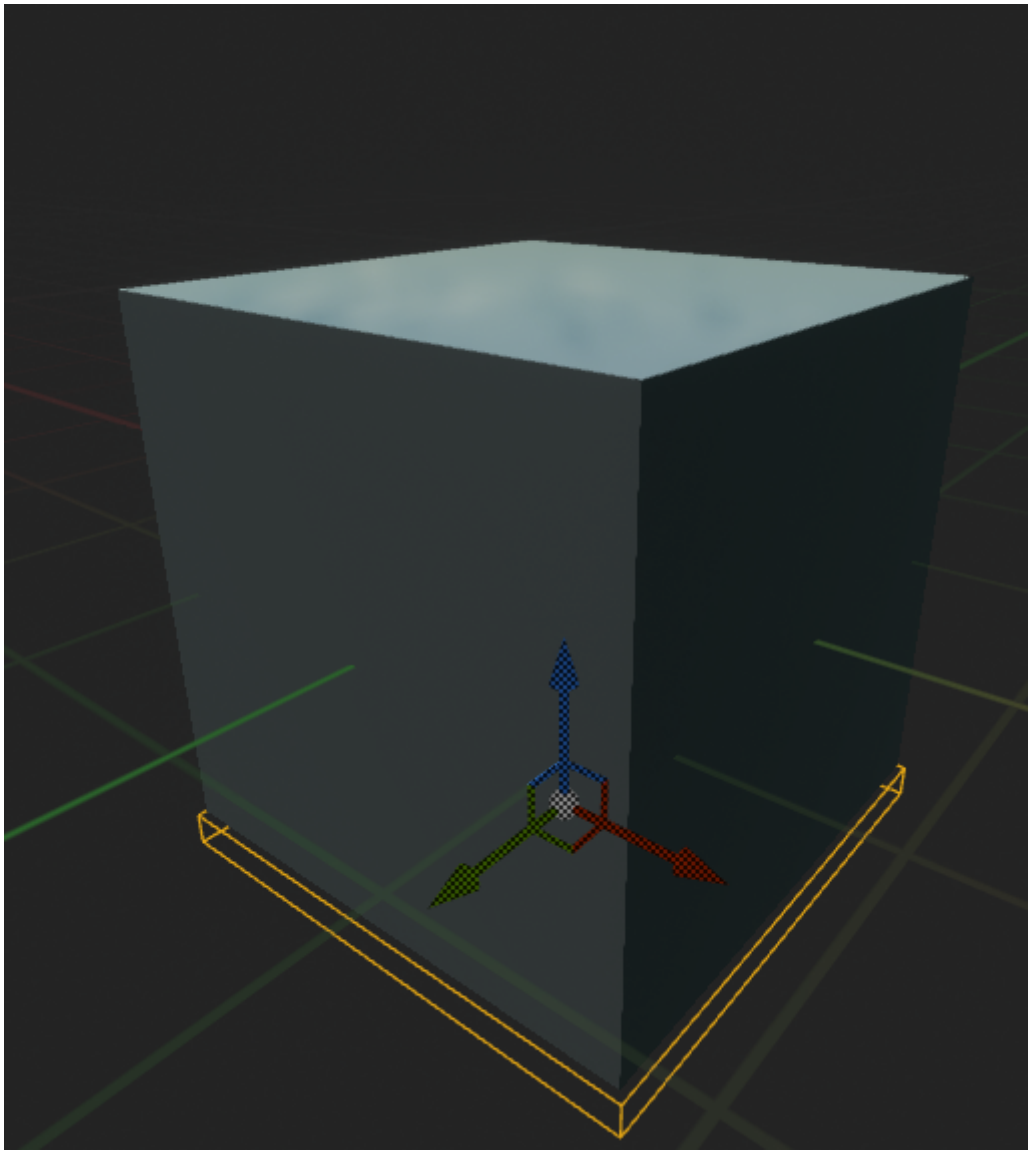
What’s actually going on here? As you may remember, we defined how we determine our keyboard input at some previous step. Here we just move each actor with the pointed details over their local Y-axis (for the camera view, it’s actually moving to the left if Y-axis is > 0 and moving to the right otherwise). So, depending on the direction we obtained from the ‘Moving’ detail we move an actor across the scene. After doing so, we disable the Moving detail, so it won’t be moving to the side anymore. Good. But what will happen when the player presses a D or a A key a second time? Do you remember the checkbox we pinned? The ‘Reuse Disabled’ in fact means that if the ‘Subjective’ has a disabled detail then the function will enable it and return as its output instead of creating a new one. So now you can run the game and check if it works. Use A & D keys to move the box around.

- Also make a few additional changes and you’ll also be able to jump.

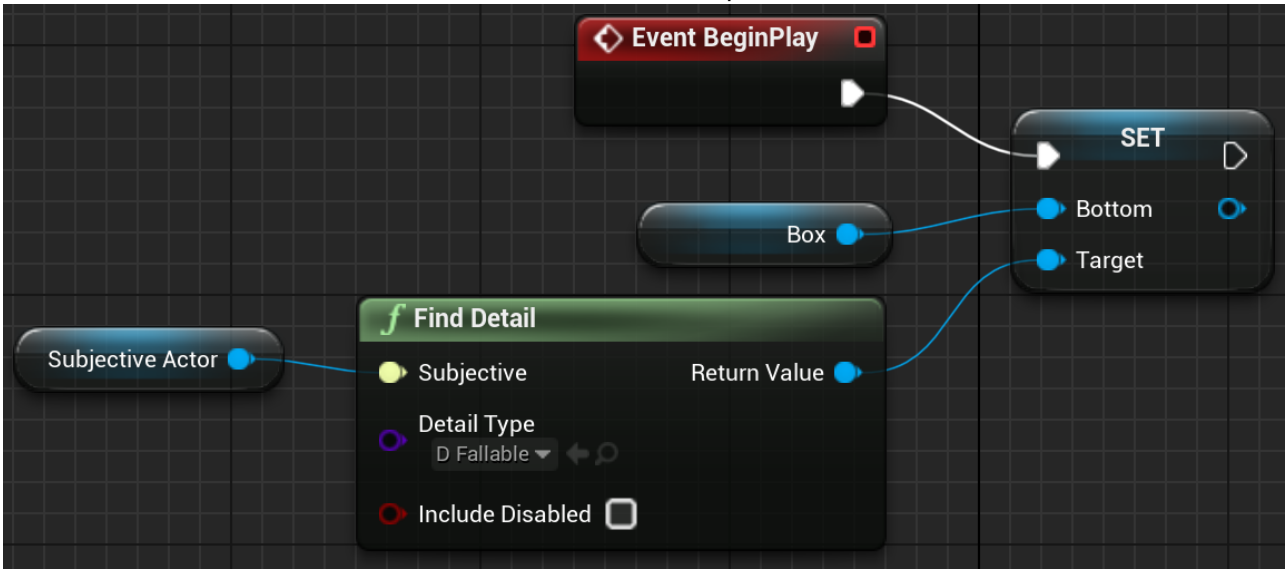


But the box isn't falling at all, because we haven't implemented the necessary logic in our Mechanism. So let's do that!

- Go back to 'BP_MainPawn' and a new 'BoxCollision' component to the 'DefaultSceneRoot'. Use the following transform:
  - Location: `(X=0.000000,Y=0.000000,Z=-13.5)`
  - Scale: `(X=0.400000,Y=0.400000,Z=0.025000)`. After that in the 'Collision' section select the 'OverlappAll' collision preset. In the 'BP_MainPawn' editor the picture should look like so:



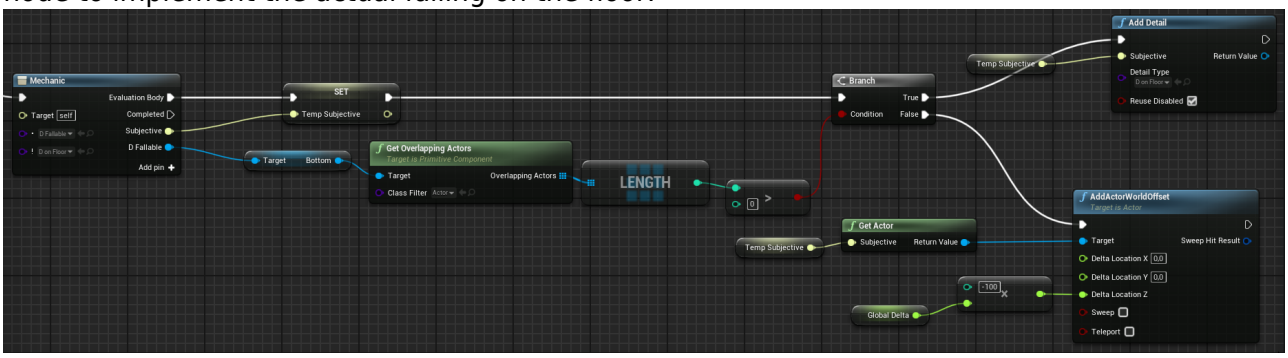
- Now navigate to the Graph and on ‘BeginPlay’ from ‘SubjectiveActor’ drag a ‘Find Detail’ function and set the bottom variable to the box of the pawn:



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

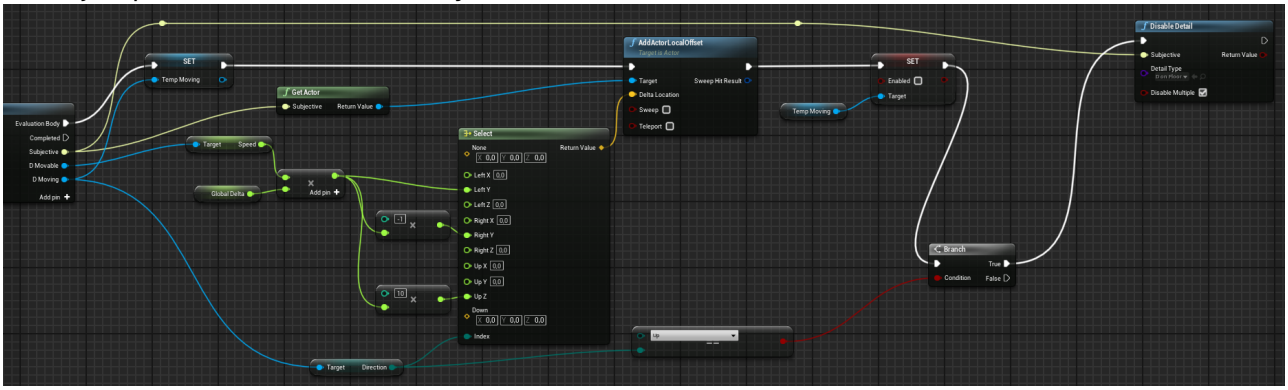
- Navigate to the level map, select ‘Floor’→‘StaticMeshActor’ and inside its details panel find the property ‘Generate Overlap Events’ and also turn it on.
- In the BP Editor of ‘BP_Mechanism’ create the next logic block by dragging from the Sequence

node to implement the actual falling on the floor:



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

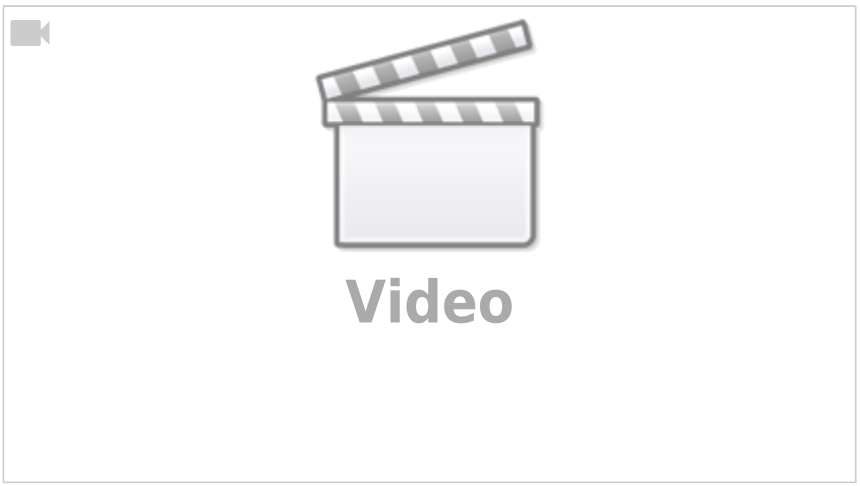
- One last thing — to begin the falling process we need to disable the ‘On Floor’ detail once the Pawn jumps. You can do it this way:



If the ‘Direction’ is ‘Up’ then we disable the detail by using the Apparatus function. Now you can both jump and fall. Just like in your real life, unless you’re living on the moon or something.

Results

Anyways, that’s it for this tutorial and the result should look similar to:



Conclusion

Apparatus is a really capable plugin for our beloved **UE**. It provides us with a bunch of new

programming principles and techniques. Those are usually called data-driven since we are more thinking detail-wise than class-wise. You can use our innovative tool in your own game production pipeline and extend its capabilities even further by declaring and implementing your own C++ classes, adhering to the necessary Apparatus interfaces.

The vast functionality of this plugin can’t be easily demonstrated on a tiny tutorial like this one. The main purpose of this article is exactly to introduce the beginners to the ECS approach in general and Apparatus in particular. Check our the following links also and don’t hesitate to ask any of your questions online on our TurboTalk forums and/or Discord server. We are eager to help and your are more then welcome to ask all sorts of questions!

Links

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

From:
http://turbanov.ru/wiki/ - **Turbopedia**

Permanent link:
http://turbanov.ru/wiki/ru/toolworks/docs/apparatus/beginner?rev=1618487943

Last update: **2021/04/15 11:59**