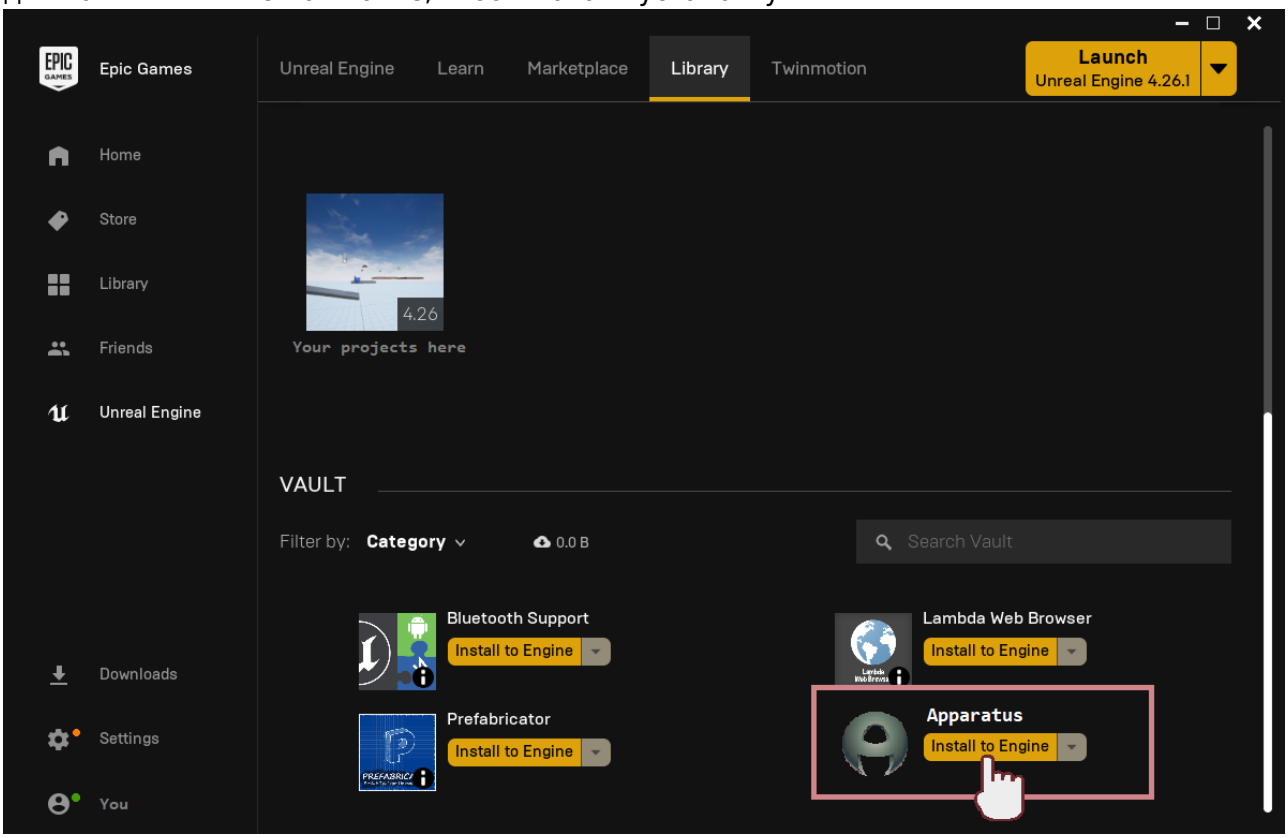
Apparatus: Введение

В этом непродолжительном уроке мы поговорим об 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.

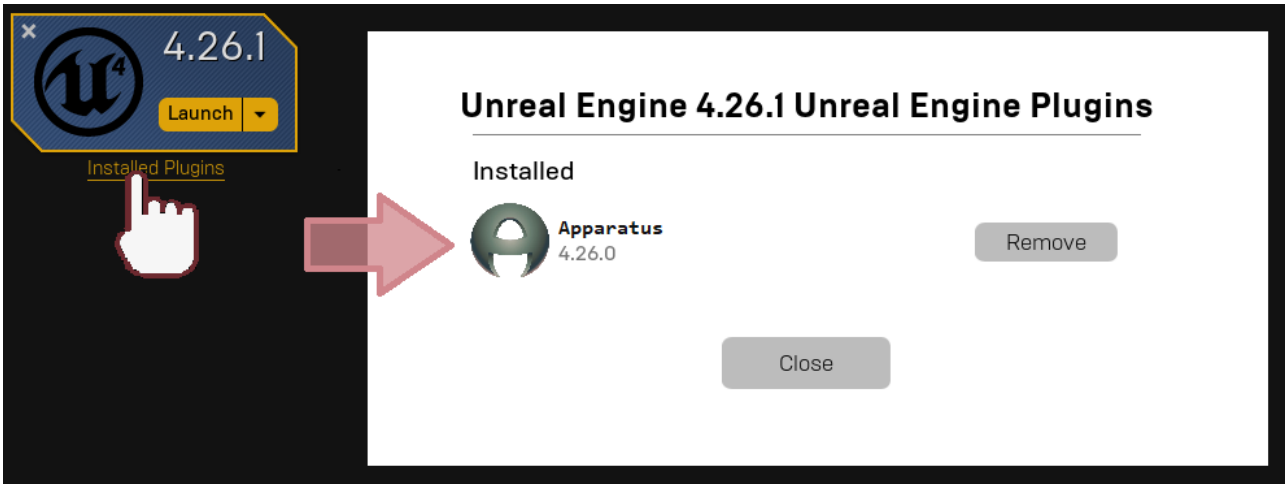
Для дальнейшего чтения необходимо понимать базовые концепции ECS-подхода. Для этого есть наша краткая справка по ECS.

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“. Кликните по кнопке, чтобы начать установку.

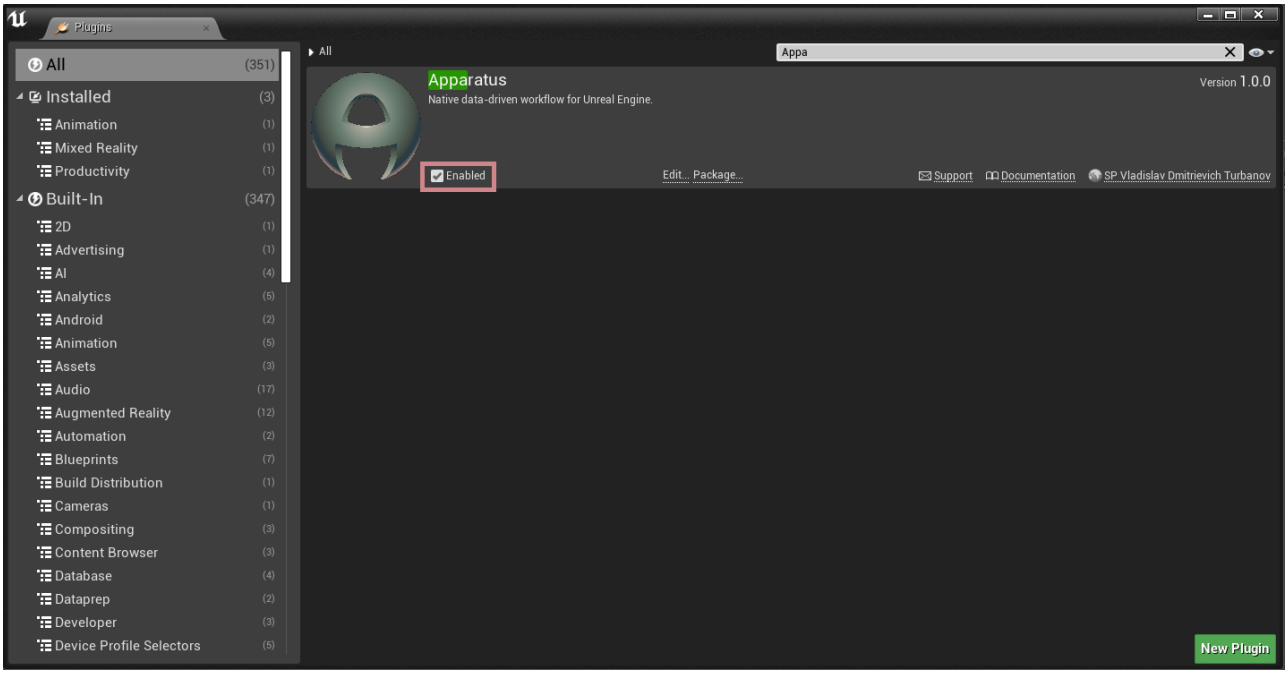


- В открывшемся окне необходимо выбрать версию Unreal Engine. Обращаем внимание, что, на настоящий момент, официально поддерживаемые версии - 4.26.1 и выше. После клика по „Установить“ подождите пару минут, пока загрузчик встроит код плагина в движок. Когда установка завершится, можно проверить её успешность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



Создание проекта

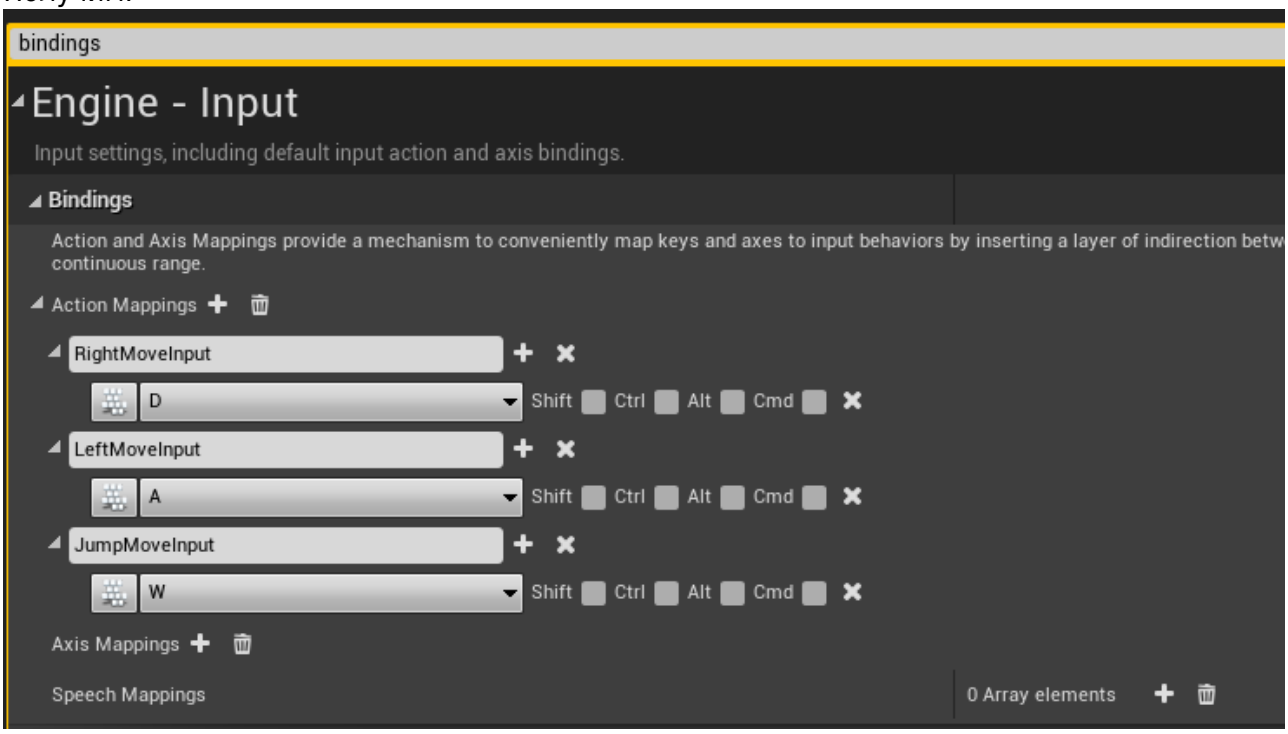
- Теперь надо создать новый проект. Выберете пустой шаблон, так как в этом уроке мы настроим всё с нуля. Остальные опции можно оставить без изменений. Мы назовём модельную игру «ApparatusLearn», ну а вы можете использовать любое другое имя.
- Когда проект создался и открылся, не лишним будет проверить, 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 опция „Enabled“ включена:



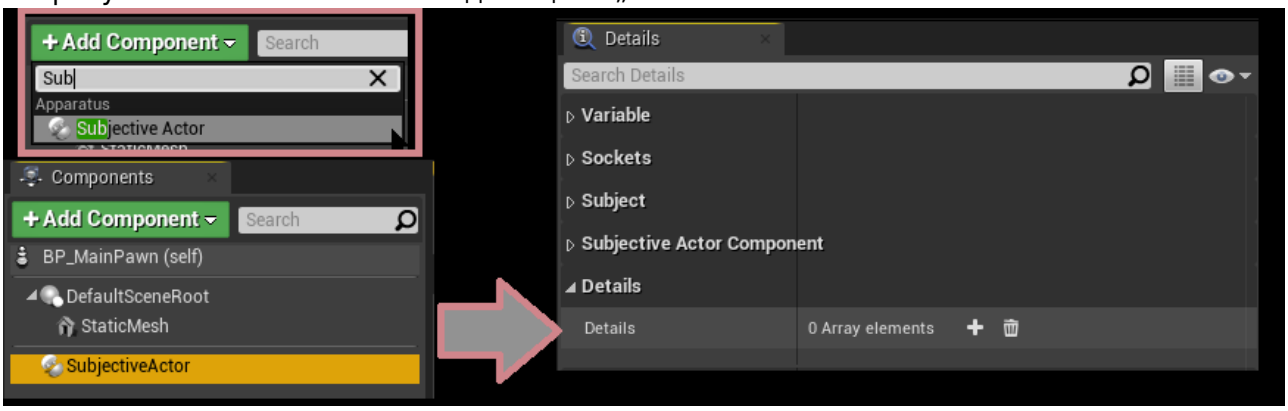
Начало работы с плагином

- Перво-наперво нам предстоит добавить привязку клавиш, чтобы понимать, когда добавлять необходимые детали к Actor-у. Чтобы это сделать, перейдём в „Edit“→„Project Settings“ и напечатаем „bindings“. Найдём секцию „Action Mappings“ и в неё добавим следующие клавиши:
  - „RightMoveInput“ – D на клавиатуре;
  - „LeftMoveInput“ – A на клавиатуре;

    - „JumpMoveInput“ – клавиша W.
- Получим:

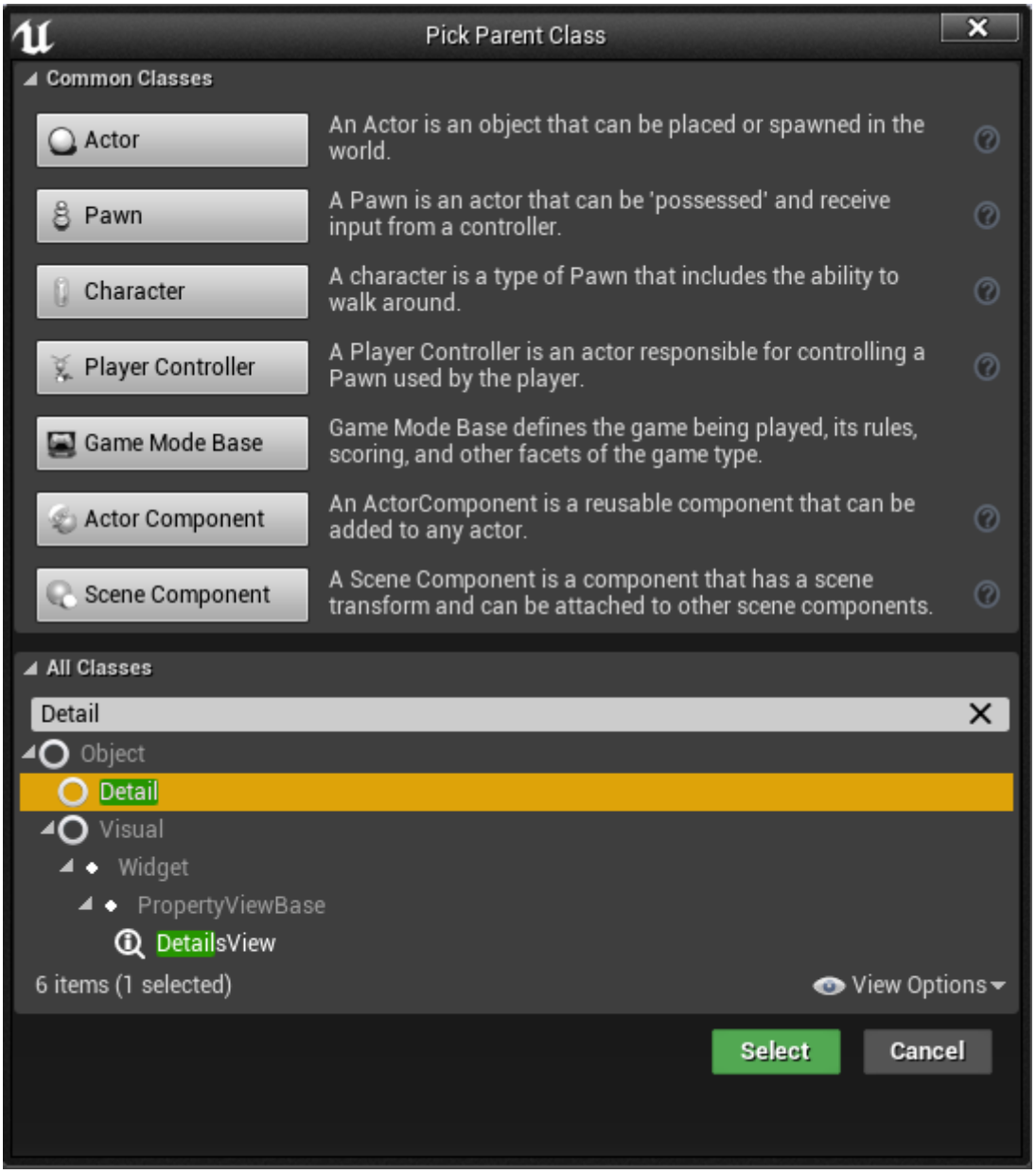


- Создадим новую „пешку“ (Pawn Blueprint). Для этого нажмём зелёную кнопку „Add/Import“ в Content-браузере. Выберем „Blueprint Class“→„Pawn“ и дадим ему имя „BP_MainPawn“ (подробнее про именование ассетов можно посмотреть в стиль-руководстве). После создания нового блупринт-класса, прожмите Ctrl+S, чтобы сохранить только что созданный объект. Теперь двойным нажатием по иконке „BP_MainPawn“ откроем редактор блупринтов (если вы в первый раз сталкиваетесь с Content-браузером, советуем пролистать официальную документацию). 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 во Viewport и к списку компонентов пешки добавим „StaticMesh“. В панели деталей справа для „Static Mesh“-свойства выберем „Cube“ mesh-ассет. Чтобы пешка выглядела более приятной, можно для „Element 0“ свойства выбрать материал „BrushedMetal“. А сейчас, пожалуйста, 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 потребуются только свойства под секцией „Details“:

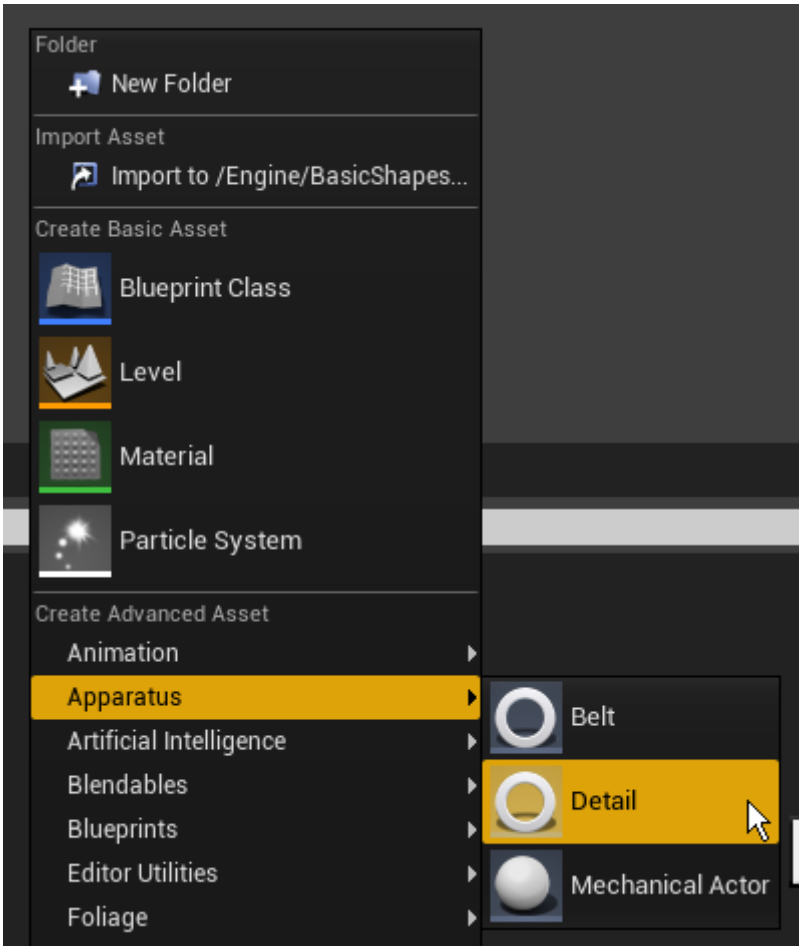


Как вы уже догадались, детали мы будем настраивать именно здесь.

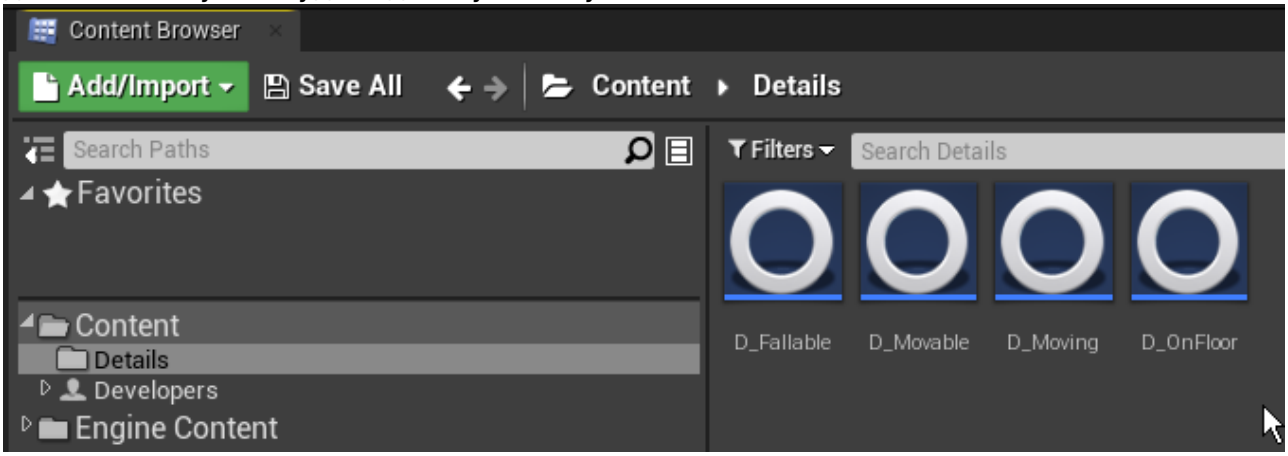
- Ctrl+Shift+S чтобы сохранить всё; пожалуйста, скомпилируйте BP. Вновь откройте Content-браузер. Здесь мы создадим новый BP, но на этот раз раскроем панель „All classes“ и найдём там класс „Detail“:



Есть также возможность создавать детали через „Create Advanced Asset“ секцию в меню Content-браузера (доступ по правой кнопки мыши):



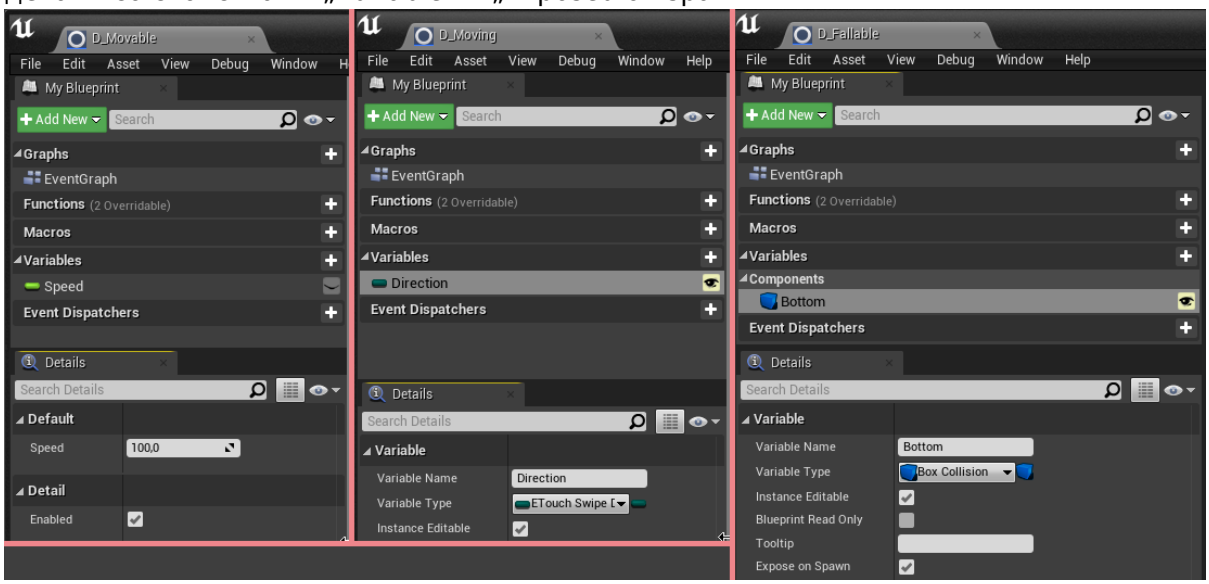
- Вообще говоря, Вы можете создать сколько угодно деталей, но для нужд этого туториала пригодятся следующие:
  - `D_Moveable`,
  - `D_Moving`,
  - `D_OnFloor`,
  - `D_Fallable`.
- Чтобы поддержать организованность проекта, переместите все созданные классы деталей в какую-нибудь отдельную папку „Details“.



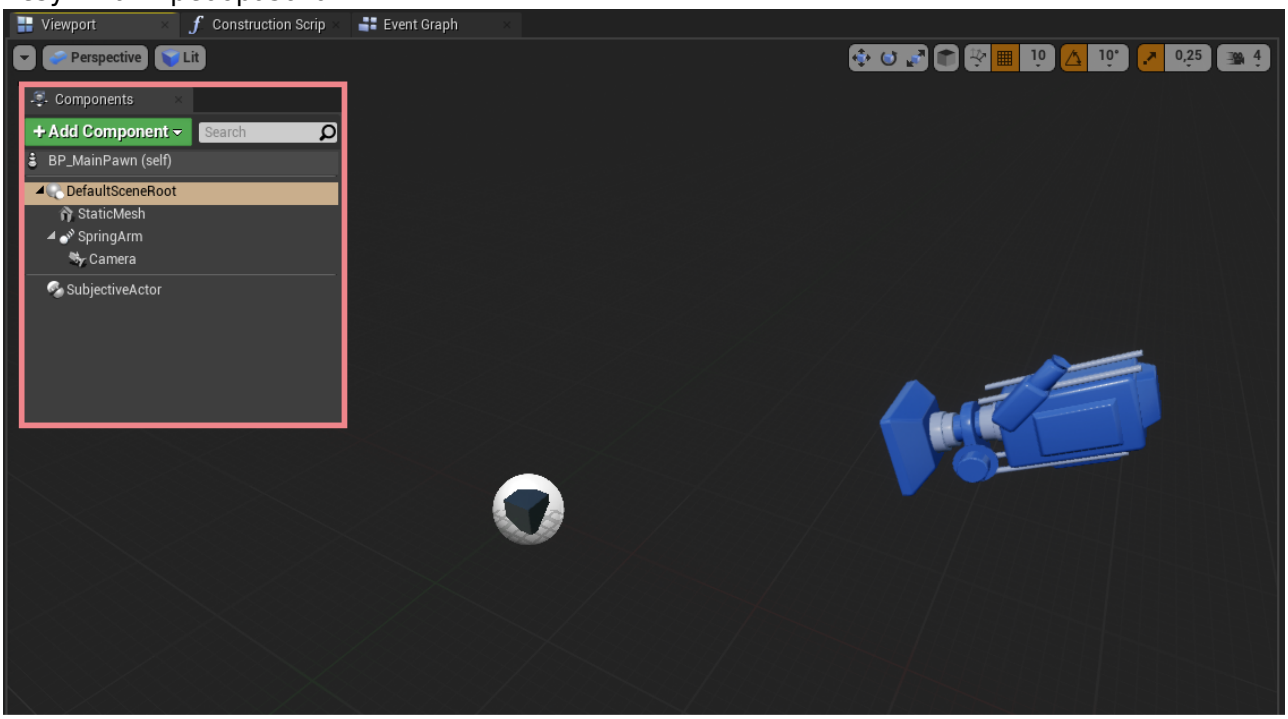
Откройте любую деталь в BP-редакторе. Несложно видеть, что все детали в Apparatus - это на самом деле обычные BP-классы, где можно по собственному усмотрению объявлять переменные, заводить макросы и функции. Так же плагин предоставляет две перегружаемые функции (override events): „Activated“ и „Deactivated“,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

- Float „Speed“ в `D_Moveable` с значением по умолчанию `100.0`.

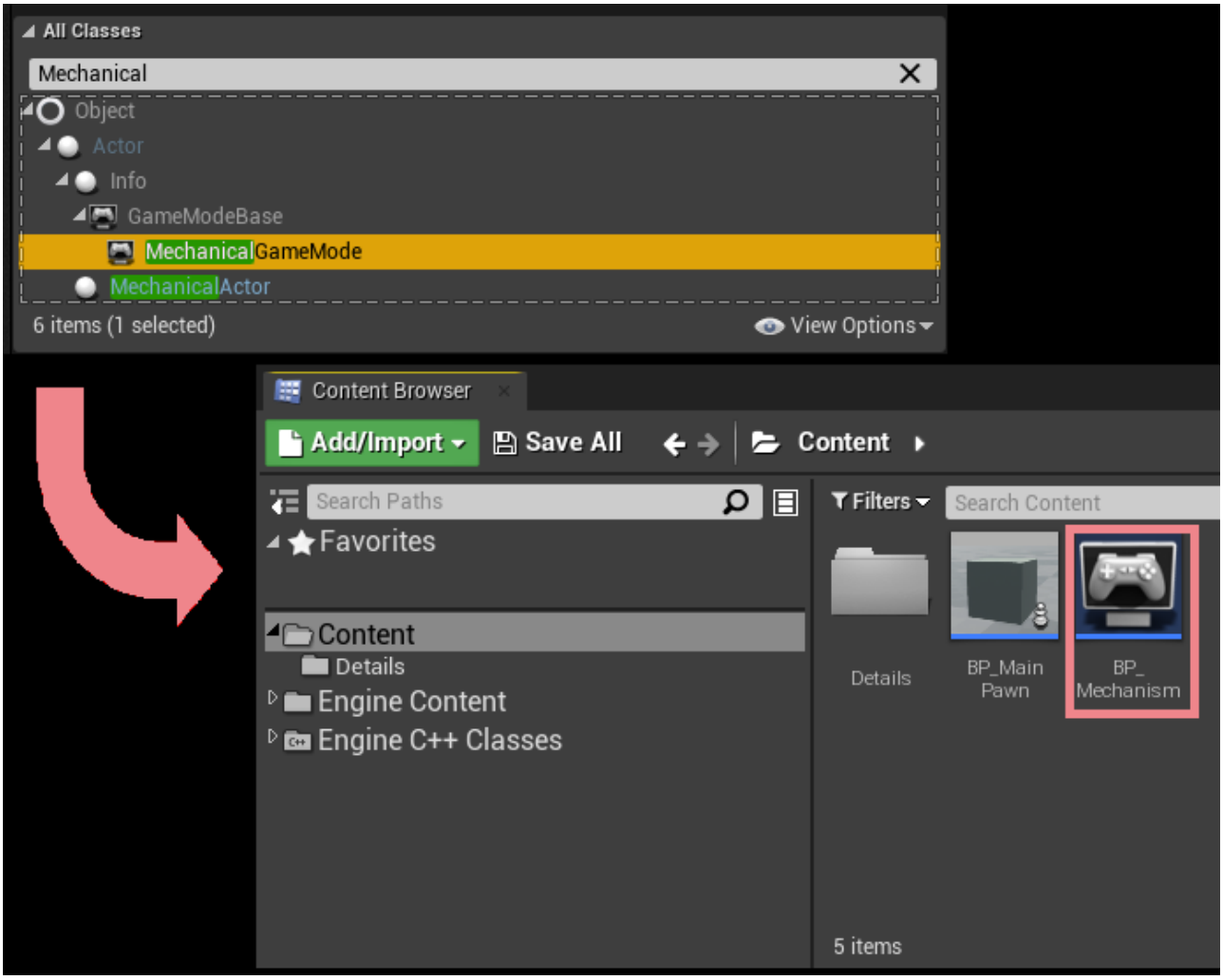
- Перечисление `ETouch Swipe Direction` с именем `Direction` со свойством „Editable“ в `D_Moving`.
- Ссылку типа `Box Collision Object Reference` с именем „Bottom“ к `D_Fallable` детали со свойствами „Editable“ и „Exposed on spawn“.



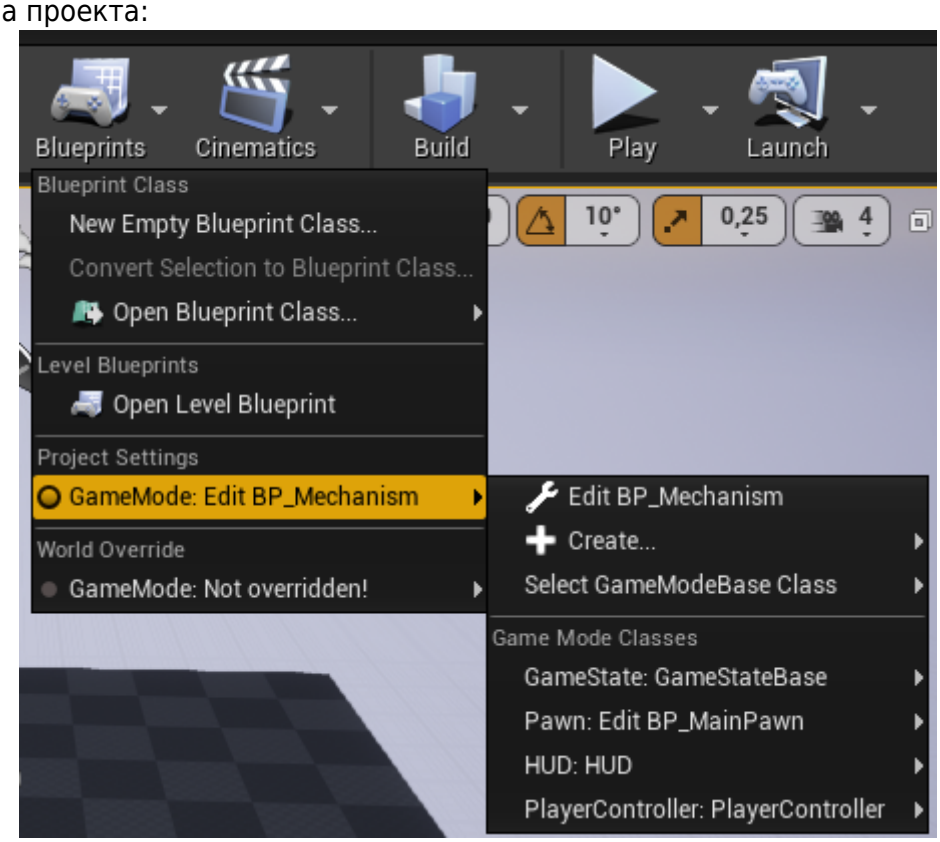
- Перейдя обратно к классу „BP_MainPawn“, сделаем куб немного поменьше (например, скопируйте эту строчку вместе с скобками: `(0.25, 0.25, 0.25)` - и, щёлкнув правой кнопкой мыши по вектору scale, вставьте скопированное значение; если всё сработает правильно, значения компонент установятся в надлежащие 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 преобразований:



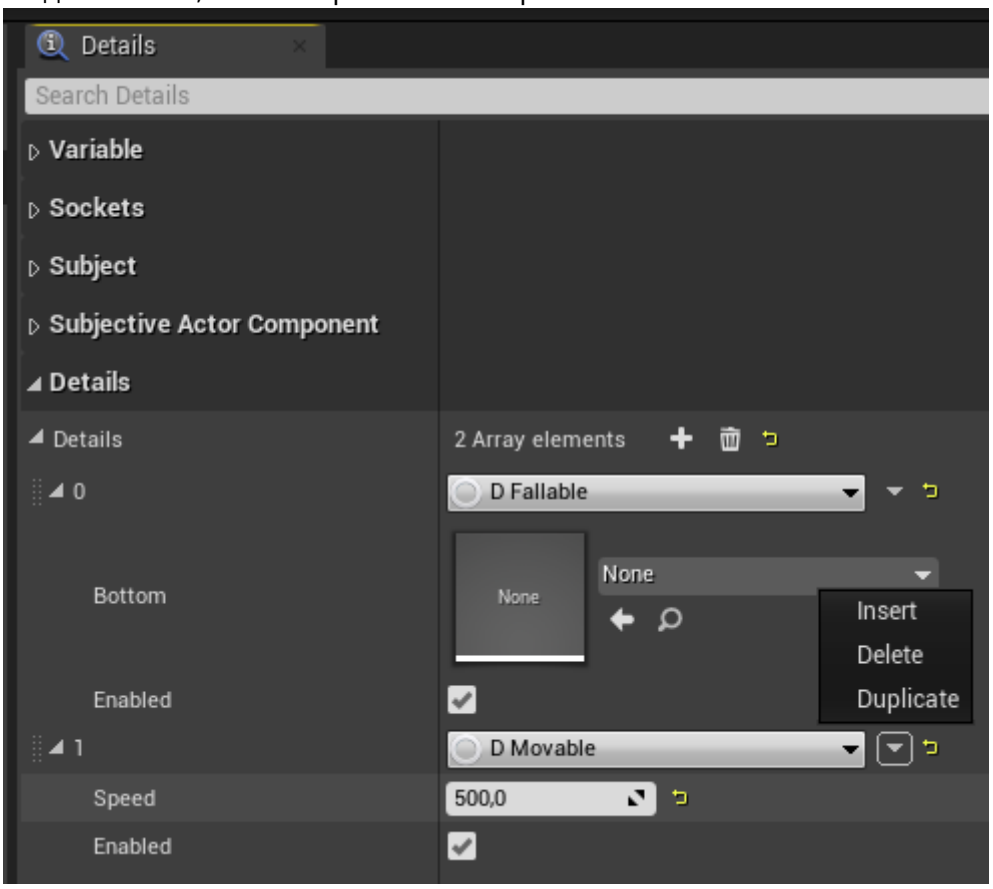
- Создадим новый „GameMode“ (или „GameModeBase“, - и тот и другой вариант приемлем) наследовавшись от „MechanicalGameMode“ (соответственно - „MechanicalGameModeBase“). Новый класс назовём „BP_Mechanism“. Примерно так:



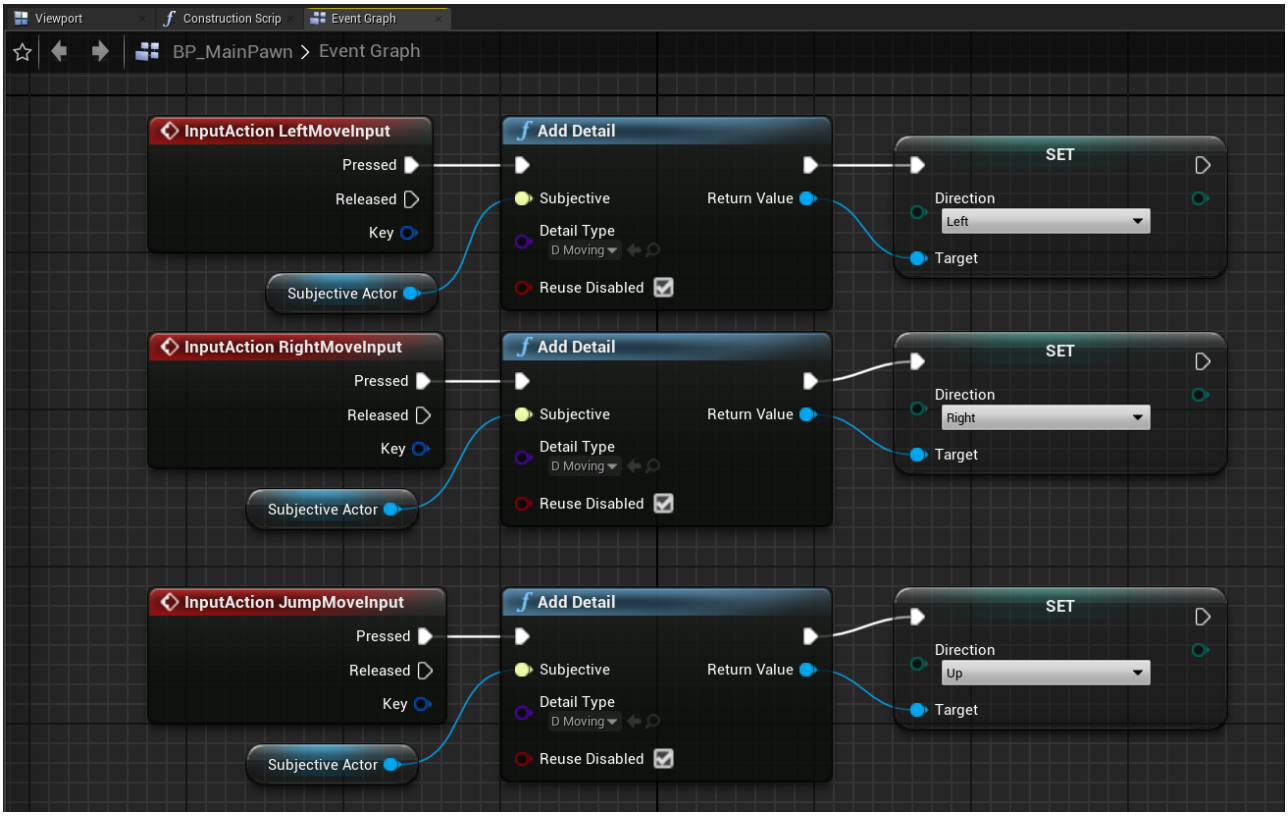
- Откройте, пожалуйста, „BP_Mechanism“ в редакторе и в панели деталей установите „Default pawn class“ в „BP_MainPawn“. Далее перейдём в настройки уровня: „Blueprints“→„Project Settings : GameMode“ и выберем „BP_Mechanism“ в качестве главного GameMode“а проекта:



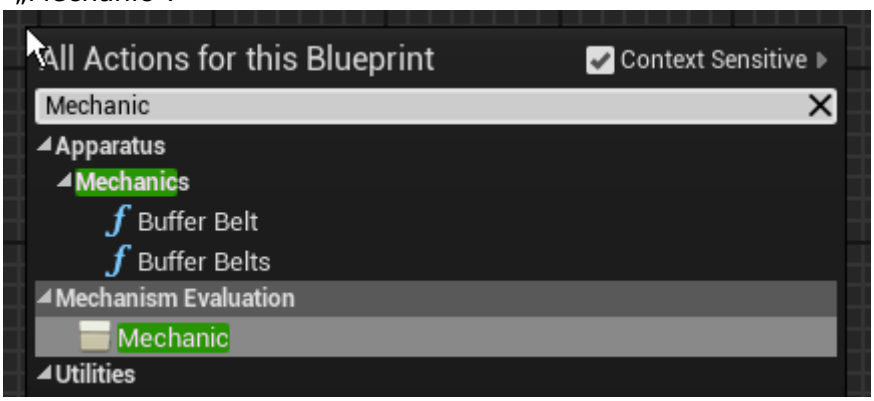
- Теперь, если запустить игру, можно видеть, что камера работает и «пешка» спавнится; но куб не двигается по нажатию на A, D или W. Исправим это. Найдём „BP_MainPawn“ в редакторе BP и в списке Actor-компонентов выберем „SubjectiveActor“, чтобы его свойства отобразились на панели деталей справа. В этой панели находим свойство „Details“ и к нему при помощи кнопки + добавляем новые детали и выбираем их типы. Нам понадобятся два класса, как изображено на скриншоте:



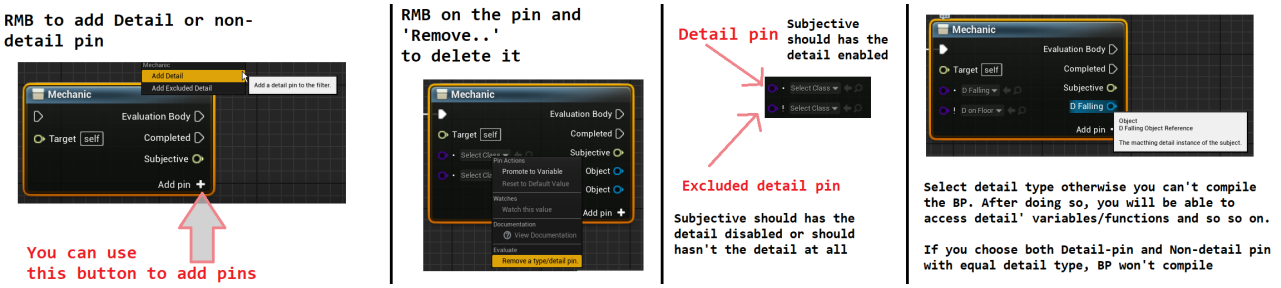
- Теперь Вы видите, что добавлять и удалять детали можно очень просто через настройки Actor компонента, причём совершенно не важно, в каком порядке они добавлены, в каком порядке в списке отображаются. Вы также можете видеть открытые переменные деталей и менять их значения по умолчанию. Заметим, что 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 Начала работы с плагином**). Создадим также 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 полям. 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:



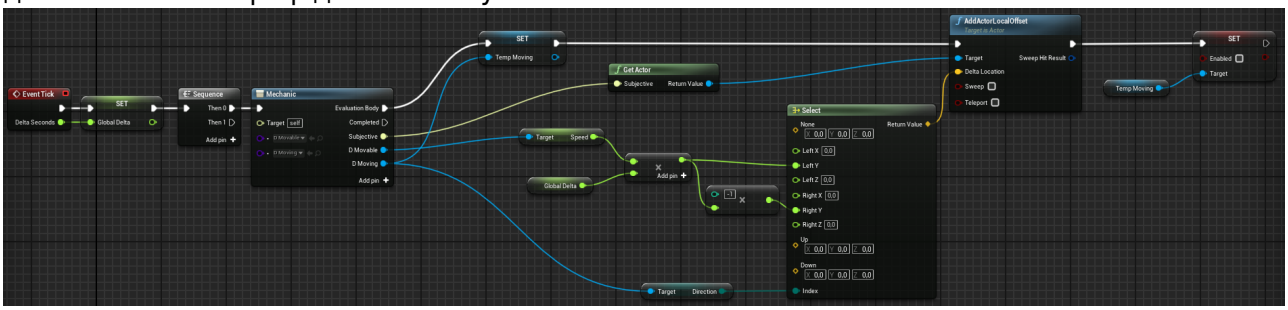
- Всё, что нам осталось сделать, - это реализовать игровые механики в нашем „GameMode“-е. Итак, откроем редактор блупринтов „BP_Mechanism“а и в графе нод удалим все события, кроме „Event Tick“. Для удобства преобразуем `Delta Seconds` в глобальную переменную `GlobalDelta`. Теперь нам необходимо интегрироваться 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 вытяните ноду „Mechanic“.



- Как нетрудно видеть, эта нода получает на вход тип „Mechanical Interface“ объекта „self“, чем и является наш „BP_Mechanism“. Эта нода выполняет обход всех сущностей с заданным набором включённых/отключённых деталей. Пустой набор означает, что итерация будет выполнена по всем subject-ам на сцене. Выход „Evaluation Body“ будет вызываться для каждой отдельной сущности, „Completed“ - когда все сущности проитерированы. Пожалуйста, нажмите правой кнопкой мыши по ноде, - в контекстном меню вы найдёте 2 последних пункта (это „Add Detail Pin“ и „Add Excluded Detail Pin“). Можете понажимать эти пункты несколько раз, и вы увидите, как при каждом таком действии к ноде добавляются новые входы с заголовками `.` и `!`. Используя эту технику и выбирая необходимые типы деталей в настройках ноды, вы определяете фильтр механики, задаёте множество сущностей, над которыми надо выполнить тот или иной код.

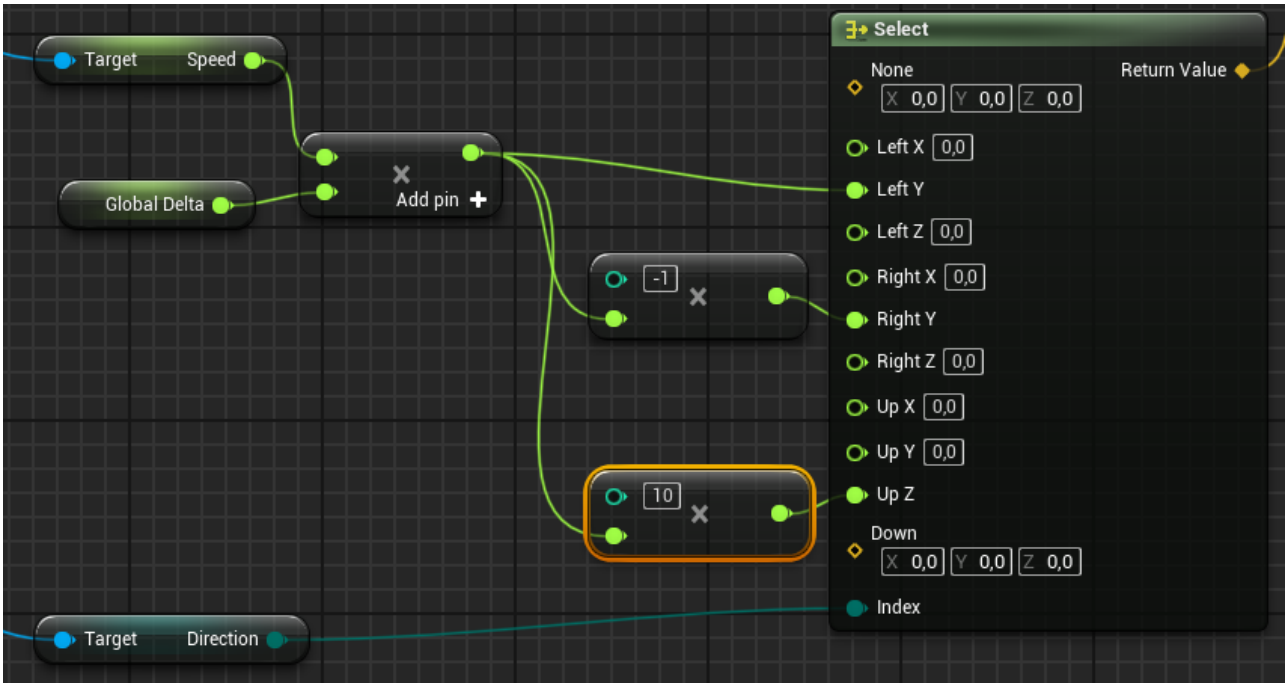


Например, пожалуйста, добавьте две .-опции и удалите остальные (правой кнопкой мыши по пункту и клик по „Remove Detail Pin“). Выберете соответствующие типы - D_Moveable and D_Moving. Преобразуйте выходное значение Moving в переменную TempMoving, чтобы блупринт-чертёж не стал походить на макароны. Из „Subjective“ выходного значения ноды „Mechanic“ вытянете „Get Actor“ pure-функцию („Subjective“ содержит ссылку на сущность, обрабатываемую в очередном вызове „Evaluation body“; а функция „Get Actor“, также предоставляемая плагином, по сущности возвращает ссылку на объект Actor-а). Из выхода „Get Actor“ вытащите, пожалуйста, „AddActorLocalOffset“ стандартную функцию UE, а из Moving выхода ноды „Mechanic“ - „Direction“ переменную, которую мы добавили в этот класс ранее. При помощи „Select“-блока мы можем легко выбирать вектор смещения отвечающий входному направлению движения, полученному из поля детали. В блоке, пожалуйста, разбейте „Left“ & „Right“ вектора по компонентам и заполните Y-компоненту случая „Left“ значением, полученным умножением переменной „Speed“ из детали „Moveable“ и глобальной переменной „GlobalDelta“. Для „Right“-случая воспользуемся тем же значением, но взятым с обратным знаком. После ноды „AddActorLocalOffset“ нужно деактивировать деталь „TempMoving“, чтобы наш куб не двигался в сторону бесконечно. Для этого достаточно изменить значение „Enabled“ детали на false. Граф должен получиться таким:



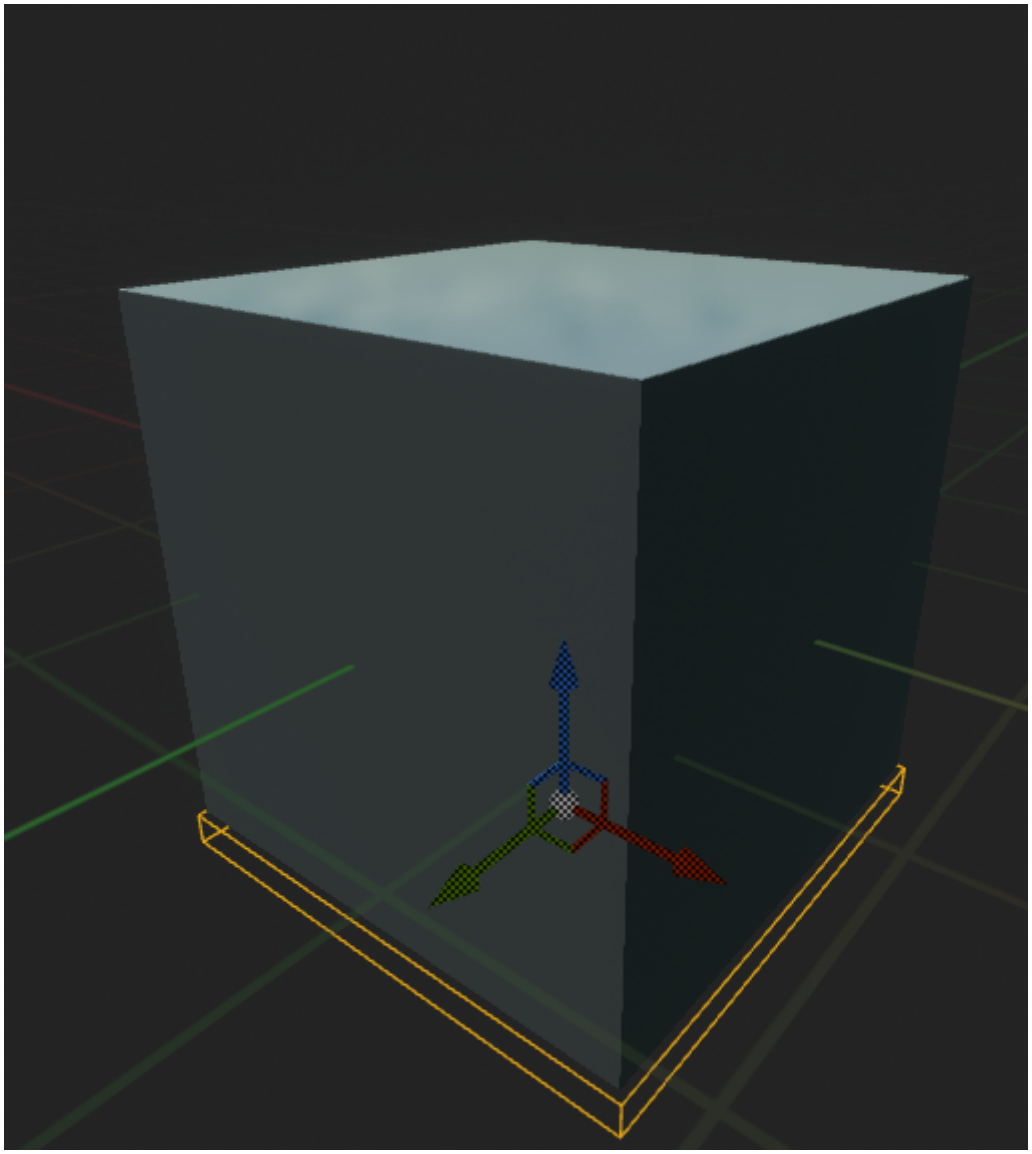
Попробуем разобраться, что же здесь происходит. Как вы помните, мы прежде описали, как нажатие клавиш игроком отражается на нашей игровой логике (добавлением деталей D_Moving). Здесь мы просто проходимся по всем сущностям с вышеупомянутыми D_Moving и D_Movable (в число которых входит только наша пешка) и для каждой такой сущности выполняем простые действия. В зависимости от направления мы сдвигаем Actor“а по Y-оси (если смотреть со стороны камеры, это как раз и будет направление влево, если смещение положительное, и вправо, если отрицательное). После сдвига выполняется отключение детали D_Moving, чтобы на следующем tick-е не обрабатывать пешку вновь. Когда игрок нажмёт клавишу D или A во второй раз, произойдёт следующее: помните checkbox-сы, которые мы устанавливали ранее? „Reuse Disabled“ на самом деле означает, что в случае, если сущность имеет выключенную деталь одноимённого типа, то выключенная деталь активируется и возвращается как выходное значение функции „Add detail“; вместо того, чтобы создавать и активировать новую, активируется и возвращается выключенная. Итак, вы можете запустить игру и убедиться в том, что всё работает. Используйте клавиши A или D, чтобы перемещать куб по сцене.

- После небольших изменений получаем возможность прыгать.

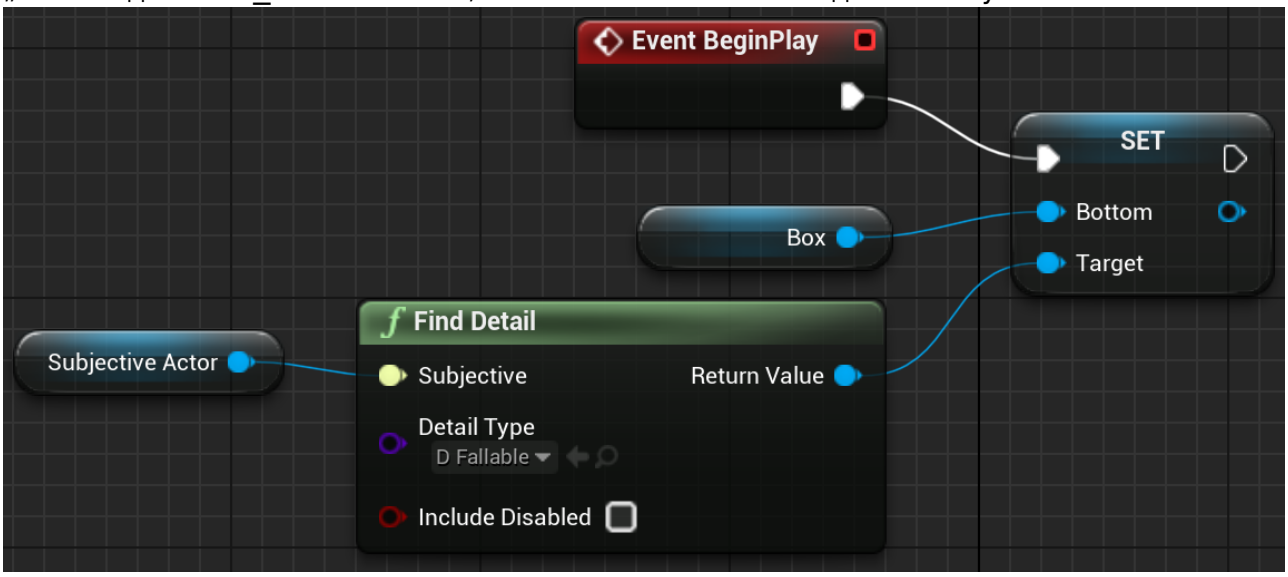


Но коробка не падает, потому что мы не реализовали отдельную для этого логику в механизме. Давайте сделаем это!

- Перейдём обратно в редактор „BP_MainPawn“ и добавим новый компонент `BoxCollision` к „DefaultSceneRoot“. Используем следующие transform-вектора:
  - Location: `(X=0.000000,Y=0.000000,Z=-13.5)`
  - Scale: `(X=0.400000,Y=0.400000,Z=0.025000)`. После этого в „Collision“ секции справа в панели деталей выберем „OverlappAll“ предустановку. Картина должна выглядеть так:



- Пожалуйста, перейдите в Graph и по событию „BeginPlay“ используя компонент „SubjectiveActor“ добавьте функцию плагина „Find Detail“ и установите переменную „Botton“ детали `D_Fallable` так, чтобы она ссылалась на добавленную коллизию:

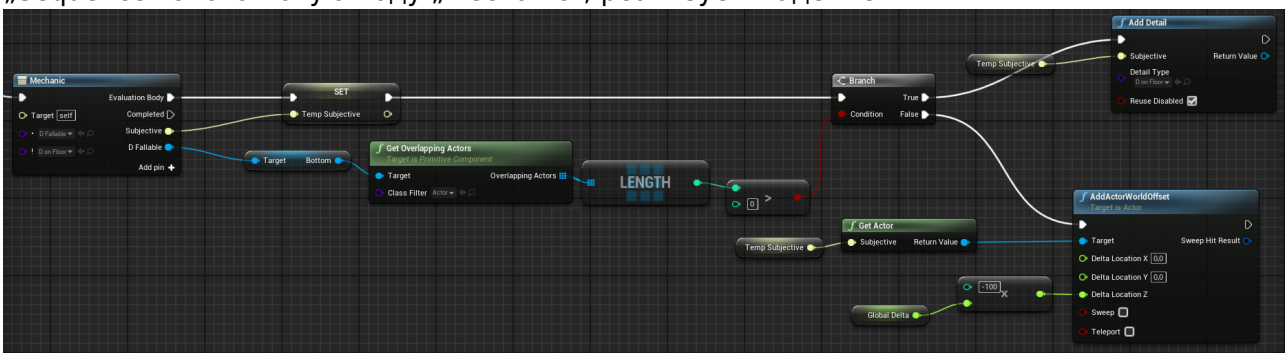


Здесь мы уверены, что сущность будет иметь данную деталь, но вы должны понимать ситуации, когда выходное значение „Find detail“ должно быть проверено перед использованием.

- Перейдя в карту уровня, выберете пол („Floor“-объект на сцене) и, пожалуйста, найдите

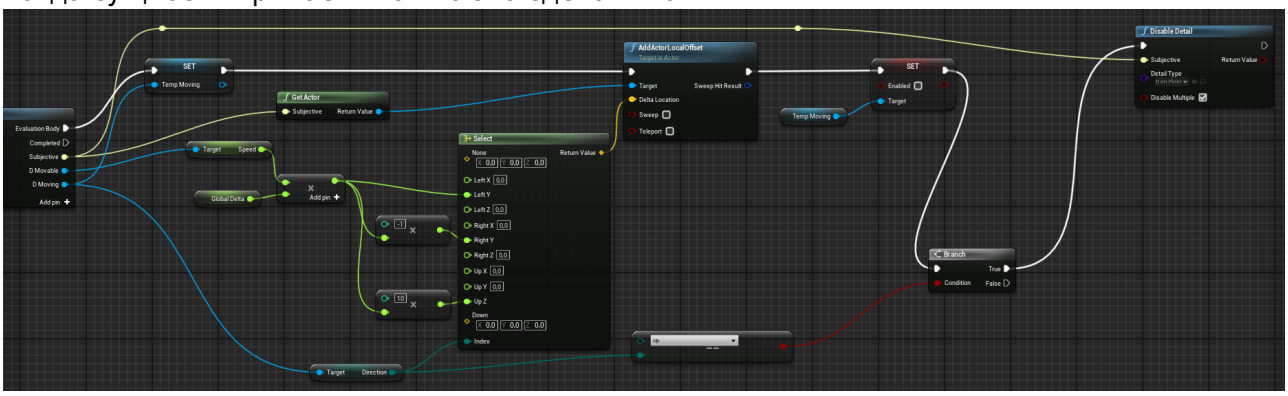
настройки его компонента „StaticMeshActor“. Среди свойств есть пункт ‘Generate Overlap Events’, который необходимо включить.

- В BP-редакторе класса „BP_Mechanism“ создадим ещё немного логики, вытягивая из „Sequence“-блока новую ноду „Mechanic“; реализуем падение:



Если сущность имеет деталь `D_Fallable` и одновременно деталь `D_OnFloor` выключена и вместе с тем дно коробки пересекается с некоторым Actor-ом, то мы устанавливаем деталь `D_OnFloor`, если же дно коробки не пересекается - значит, текущую сущность нужно сдвинуть вниз, как бы симулируя равномерное падение.

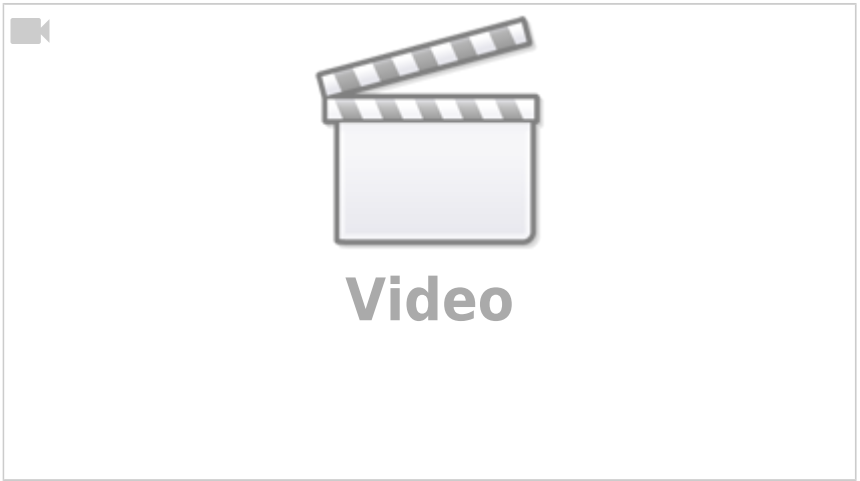
- Последнее - чтобы начать процесс падения, нам надо выключить деталь `D_OnFloor`, когда сущность прыгает. Можно это сделать так:



Если направление - Вверх, то мы выключаем деталь, опять же используя встроенную функцию плагина. Теперь можно и прыгать и падать.

Результат

На этом всё, результат проделанной работы должен выглядеть примерно так:



Итоговый проект на github, ветка ApparatusLearn#1

Вывод

Apparatus - это очень удобный и многофункциональный плагин, даже больше чем плагин - это полноценный дата-ориентированный фреймворк. Он предоставляет набор новых принципов разработки и техник проектирования. Эти техники обычно называются data-driven, так как мы по большей части мыслим уже деталями, а не классами или наследованием. Вы можете использовать наш инновационный инструмент в своих собственных разработках игр и расширять свои возможности, реализуя собственные классы C++, имплементирующие необходимые интерфейсы Apparatus“a.

Все функциональное множество этого плагина не может быть продемонстрировано в столь коротком уроке, как этот. Основной же мотив туториала состоял именно в том, чтобы ввести новичка в мир ECS и Apparatus-a в частности. Не брезгайте проверить наши ссылки на другие ресурсы и смело задавайте возникшие вопросы в TurboTalk-форуме и/или в Discord-чате. Мы обеими руками за обратную связь, готовы давать своевременную поддержку и приветствуем заинтересованность любого характера!

Ссылки

- Более сложный проект на GitHub
- Online API Reference

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

Permanent link:
http://turbanov.ru/wiki/ru/toolworks/docs/apparatus/beginner

Last update: **2022/01/04 16:06**