ООО «КРИСТА»
ОГРН 5177746037520, ИНН/КПП 7714415927/771501001
127015, г. Москва, ул. Новодмитровская, д. 2, корп. 1, эт. 7, пом. L, ком. 10

ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ «ЭКСПАНСИЯ» (ВЕРСИЯ 1.0)

МОСКВА 2024

Экспансия 1.0

Набор инструментов разработки игр Экспансия 1.0 включает следующие элементы:

1. Система локализации игра на Unreal Engine Localization System. Предназначена для управления локализацией текстов, контроля состояния, экспорта и импорта заданий на локализацию для переводчиков. Состоит из двух пакетов:
    a. плагин LocalizationSystem для Unreal Engine 5.4.4
    b. Windows приложение Localization Tracker
2. Система задания квестов QuestSystem. Предназначена для создания структуры квестов с целью последующего использования игровой логикой. Представляет собой плагин для Unreal Engine 5.4.4.
3. Система создания и проигрывания диалогов DialogueSystem. Предназначена для создания графов диалогов, написания текстов, и подключения к ним игровой логики. Также предоставляет способ проигрывания диалогов в игре. Состоит из:
    a. Плагина DialogueSystem для Unreal Engine 5.4.4.
    b. Демонстрационного проекта DialogueSystemDemo
4. Система создания и проигрывания игровых сцен MisansceneSystem. Предназначена для создания скриптовых сцен произвольной сложности в игре, включая синхронизацию различных акторов и нелинейное выполнение сцены. Представляет собой плагин для Unreal Engine 5.4.4.
5. Система игровой логики EtudeSystem. Предназначена для задания комплексной игровой логики в виде графа состояний, с помощью специализированного редактора, и исполнения этой логики в процессе игры. Представляет собой плагин для Unreal Engine 5.4.4.
6. Система генерации укрытий CoverSystem. Предназначена для автоматической генерации сплайнов и метаинформации о препятствиях в игровом уровне, которые могут являться укрытиями для игрока в игре от третьего лица. Представляет собой плагин для Unreal Engine 5.4.4.
7. Базовые зависимости для других плагинов. Не предоставляет отдельной функциональности, используется как зависимость другими элементами пакета. Представляет собой пакет из нескольких плагинов для Unreal Engine 5.4.4, используется как зависимость другими инструментами в наборе.
8. Инструмент создания мешей головы DNA Calibration Tool. Предназначается для создания новых мешей головы методом комбинирования исходных, созданных с помощью системы Metahuman. Представляет собой плагин для Unreal Engine 5.4.4.
9. Инструмент отладки коллизий Collision Detector. Предназначена для поиска пересечений коллизий при редактировании уровней. Представляет собой плагин для Unreal Engine 5.4.4.

10. Инструмент сдвига акторов Compensator. Предназначена для массового редактирования акторов в случае, если начало координат исходного объекта изменилось. Представляет собой плагин для Unreal Engine 5.4.4.
11. Инструмент автозамены AutomaticBlueprints. Предназначен для автоматизации создания блюпринтов из статических мешей непосредственно в месте использования. Представляет собой плагин для Unreal Engine 5.4.4.
12. Инструмент автоматического слияния VersionManager. Предназначена для автоматического переноса коммитов между ветками в системе контроля версий, основываясь на информации и трекера задач Jira. Представляет собой Windows приложение. Требует для работы системы контроля версий Git и трекера задач Jira.
13. Инструмент отправки сообщений об ошибках Krista Bugreport. Предназначен для сбора информации об ошибках из редактора Unreal или из игры, с дополнительной контекстной информацией. Представляет собой плагин для Unreal Engine 5.4.4, требует для работы настроенный и запущенный сервер Siren
14. Сервис Siren, предназначенный для автоматизации процесса обработки баг-репортов и их интеграции с системой управления проектами Jira.Представляет собой сервис для Ubuntu/Debian систем.
15. Система Tiamat, система управления хуками Git, разработанная для работы с множеством репозиториев на платформе GitLab. Представляет собой сервис для Ubuntu/Debian систем.
16. Remote Console - программа удаленного управления игровыми приложениями. Состоит из:
    a. Windows приложение RemoteConsole
    b. Демонстрационный проект remoteconsole-example-project для использования с движком Unity.

Установка инструментов

Для автоматической установки используется приложение Krista Installer.
Запустите приложение, в открывшемся окне выберите галочками, какие составляющие инструменты нужно установить.
Для плагинов Unreal Engine укажите путь к проекту Unreal, в который нужно установить плагины. Плагины поддерживают Unreal Engine **версии 5.4.4**. Для использования инструментария с другими версиями движка необходимо будет собирать плагины из исходного кода, что требует настроенного окружения для сборки C++ кода в Unreal (Microsoft Visual Studio или MSBuild)
Для самостоятельных приложений укажите папку, в которую они будут установлены.
Нажмите “установить”.

Ручная установка

В комплекте поставки находятся файлы архивов с каждым компонентом инструментария. Архивы необходимо распаковать в соответствующие папки:

1. Система локализации
    a. LocalizationSystem.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    b. BasePlugins.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    c. LocalizationTracker.zip в любую папку, например C:\LocalizationTracker.
    d. Для работы LocalizationTracker необходимо также установить расширение .NET runtime версии 7.0.2 по ссылке https://versionsof.net/core/7.0/7.0.2/
2. Система задания квестов QuestSystem.
    a. QuestSystem.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    b. BasePlugins.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
3. Система создания и проигрывания диалогов DialogueSystem.
    a. DialogueSystem.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    b. BasePlugins.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    c. LocalizationSystem.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    d. По желанию: DialogueSystemDemo.zip в любую папку (Например C:\Demo)
4. Система создания и проигрывания игровых сцен MisansceneSystem.
    a. MisansceneSystem.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    b. BasePlugins.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    c. По желанию: MisansceneSystemDemo.zip в любую папку (Например C:\Demo)
5. Система игровой логики EtudeSystem.
    a. EtudeSystem.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    b. BasePlugins.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    c. По желанию: EtudeSystemDemo.zip в любую папку (Например C:\Demo)
6. Система генерации укрытий CoverSystem.
    a. CoverSystem.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    b. BasePlugins.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)

7. Базовые зависимости для других плагинов.
    a. BasePlugins.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
8. Инструмент создания мешей головы DNA Calibration Tool.
    a. DnaCalibTool.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
9. Инструмент отладки коллизий Collision Detector.
    a. CollisionDetector.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
10. Инструмент изменения пивота акторов Compensator.
    a. CollisionDetector.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
11. Инструмен автозамены AutomaticBlueprints
    a. AutomaticBlueprints.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
12. Инструмент автоматического слияния VersionManager.
    a. VersionManager.zip в любую папку, например C:\VersionManager.
    b. Для работы VersionManager необходимо также установить расширение .NET runtime версии 7.0.2 по ссылке https://versionsof.net/core/7.0/7.0.2/
13. Инструмент отправки сообщений об ошибках Krista Bugreport
    a. KristaBugreport.zip в папку Plugins внутри проекта Unreal (Например C:\MyProject\Plugins)
    b. Для работы необходим установленный и запущенный сервер Siren (см. Соответствующую документацию)
14. Инструкция по установке Siren находится в Siren.docx
15. Инструкция по установке Tiamat находится в Tiamat.docx
16. Инструмент удалённого управления игровыми приложениями Remote Console
    a. RemoteConsole.zip в любую папку (например C:\RemoteConsole)
    b. Убедиться, что на компьютере установлен редактор Unity версии 2022.3.7f1
    c. Открыть в Unity проект C:\RemoteConsole\remoteconsole-example-project

Инструменты, требующие дополнительной настройки:

1. LocalizationTracker по умолчанию настроен на тестовую базу строк. Чтобы подключить его к Unreal проекту, необходимо в папке установки найти файл config.json (например C:\LocalizationTracker\config.json), и с помощью любого текстового редактора заменить в нем путь к проекту (настройка StringsFolder). Путь должен быть задан относительно той папки, в которой находится файл конфигурации, например ..\MyProject\Content\Strings.
   Если папки Strings в проекте не существует (например, потому что в проекте еще не создали ни одной строки), её необходимо создать вручную перед запуском инструмента.
2. VersionManager требует правильно настроенных Jira и Git. Подробная информация о настройке окружения - в отдельном документе VersionManager.pdf

Localization System

Система локализации текстов состоит из двух элементов: плагина LocalizationSystem для Unreal Engine и внешнего инструмента LocalizationTracker для работы с текстами и переводами. В этом документе описан только плагин.

Плагин LocalizationSystem

Плагин требует плагинов KristaMisc и KristaAssertions для работы.

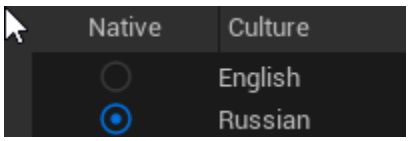
Как установить плагин в проект

1. Установить плагины KristaMisc и KristaAssertions по их инструкции.
2. Перенести директорию LocalizationSystem в директорию Plugins проекта.
3. Запустить редактор
4. Перейти в меню Edit -> Plugins
5. Найти плагин LocalizationSystem и поставить галку напротив
6. Перезапустить редактор
7. В настройках локализации (меню Tools/Localization Dashboard) добавить русскую локаль и пометить её как Native
8. В настройках сбора локализации в том же окне поставить галку Gather From Packages и хотя бы одну папку в Include Path Wildcards (типично там должна быть папка Content/*)

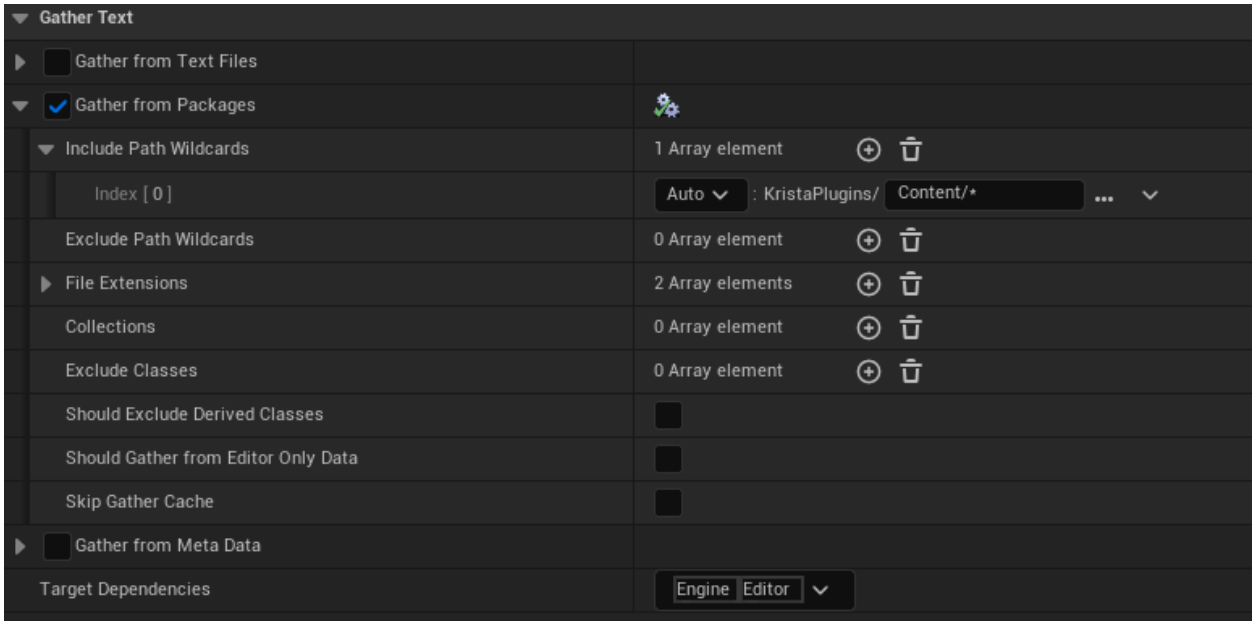
Базовые понятия

Локализуемый текст в Unreal записывается прямо внутри ассетов (или даже кода) в бинарном виде. Работать с локализацией в таком виде неудобно, поэтому все наши данные хранятся в отдельных текстовых файлах. Файлы эти лежат в папке ProjectDir\Content\Strings\ и вложенных папках - структура дублирует структуру папок в контенте. В этих файлах непосредственно сохраняются переводы текста, а также метаинформация (комментарии, трейты и т.п.)
Строки в ассетах автоматически синхронизируются с файлам при редактировании. Также существует коммандлет SyncLocalization, который проверяет и обновляет синхронизацию всех строк в проекте. Он предназначен для использования системой CI/CD для периодического вызова, чтобы исправлять возможные ошибки, внесенные одновременной работой разных пользователей, и для первичной синхронизации при включении системы в новом проекте.
Непосредственно применение локализованных текстов осуществляется встроенной системой Unreal Engine, то есть для обновления переводов в игре так же необходим вызов компиляции локализации. Для удобства он также включен в работу коммандлета SyncLocalization.

Для использования системы необходимо, чтобы в редакторе была установлена русская локаль (“ru”) в качестве нативной.



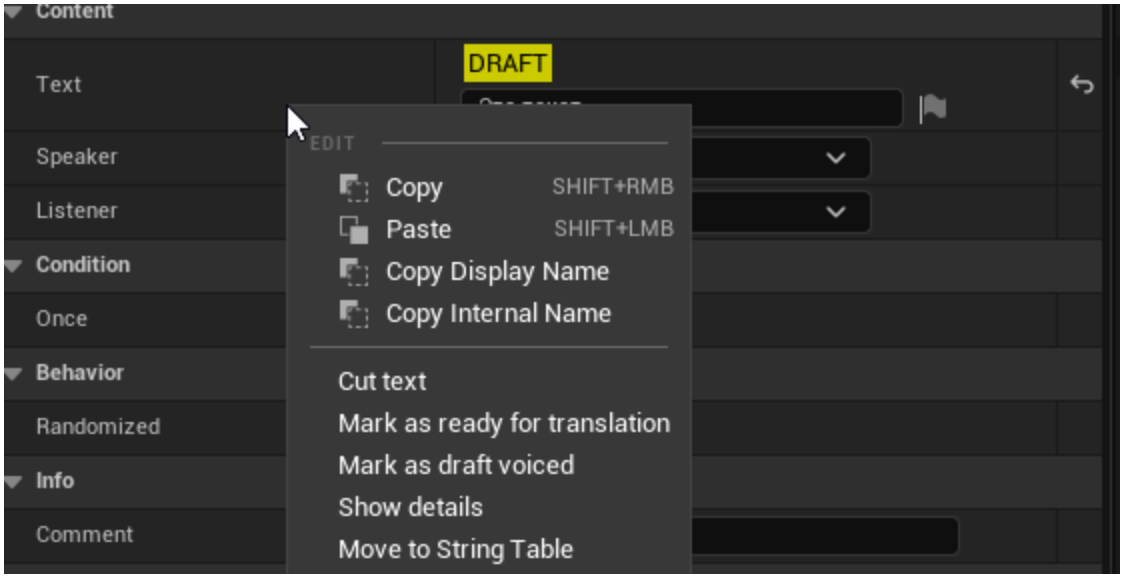
Также важна правильная настройка сборки локализации: плагин LocalizationSystem использует встроенный механизм обнаружения локализованных текстов Unreal, поэтому тексты будут работать только в тех ассетах, которые попадают в процесс Localization Gather. Пример настройки



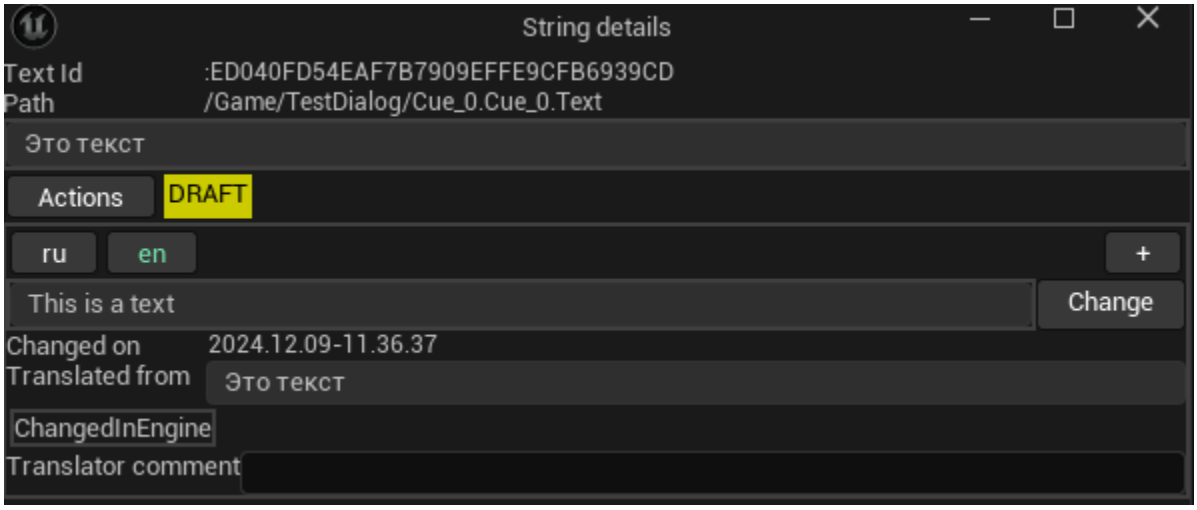
Работа с текстом в движке

Все локализуемые тексты это поля в ассетах. Технически, Unreal умеет локализовывать строки, которые упоминаются напрямую в коде или в блюпринтах, но это плохо работает с пайплайном, предполагаемым LocalizationTracker..

Все текстовые поля в ассетах показывают кастомизированный интерфейс. В нем можно увидеть или поменять статус строки и вообще все её внутреннее представление:



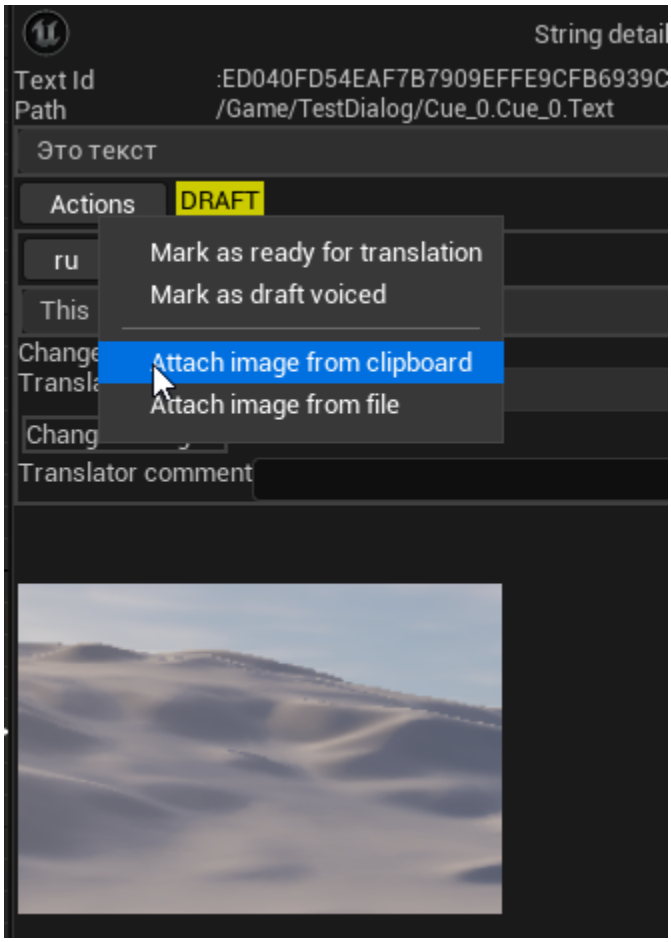
Над текстовой строкой - место с тегами, которые отображают статус строки. Также тут будет указано, если у строки еще не создан json файл, или если строка из StringTable (в этом случае её и редактировать надо в StringTable). В контекстном меню - пункты для изменения статусов, плюс Show Details, который открывает окно с подробной информацией по строке.



В окне String Details повторяются все те же статусы, но также можно увидеть трейты (это внутренние флаги, которые могут быть как у строки в целом, так и у конкретного перевода) которые собственно определяют статус. Также тут можно увидеть все существующие переводы, и их трейты. Перевод на русский язык - это тоже *перевод*, и может отличаться от того текста, который указан в ассете. В билде будет использован именно этот "перевод с русского на русский" - он автоматически синхронизируется при редактировании ассета.

Также в каждом переводе есть комментарий для переводчика. Для русского текста его настоятельно рекомендуется заполнять в движке, чтобы отдел локализации знал что с ним делать (в других локалях он нужен, чтобы перевести комментарий для тех, кто будет переводить уже не с русского)

Кнопка Actions позволяет менять статус + добавлять к строке картинку. Это важно для UI строк, где переводчику нужно видеть пример использования в игре. Приаттаченный скриншот кладется в виде файла в папку ProjectDir\Content\Localization\Attachments.



Статусы строк

Движок понимает следующие статусы:

- DRAFT. Строка черновая, переводить не надо.
- Ready for translation. Строка финальная, можно отправлять на перевод.
- Draft Voiced. У строки есть черновая озвучка.
- Final Voiced. У строки есть финальная озвучка.
- Sent to translate. Строка отправлена на перевод.
- Translated. Строка уже переведена.
- Sent to editor. Переведенная строка отправлена на редактуру.
- Edited. Строка переведена и прошла редактуру.
    - Предыдущие четыре статуса могут быть разные у разных языков; в редакторе отображается самый сильный из заданных (т.е. если английский отредактирован, а немецкий только отправлен, строка будет помечена Edited)

Операции со строками

Дополнительные операции со строками:

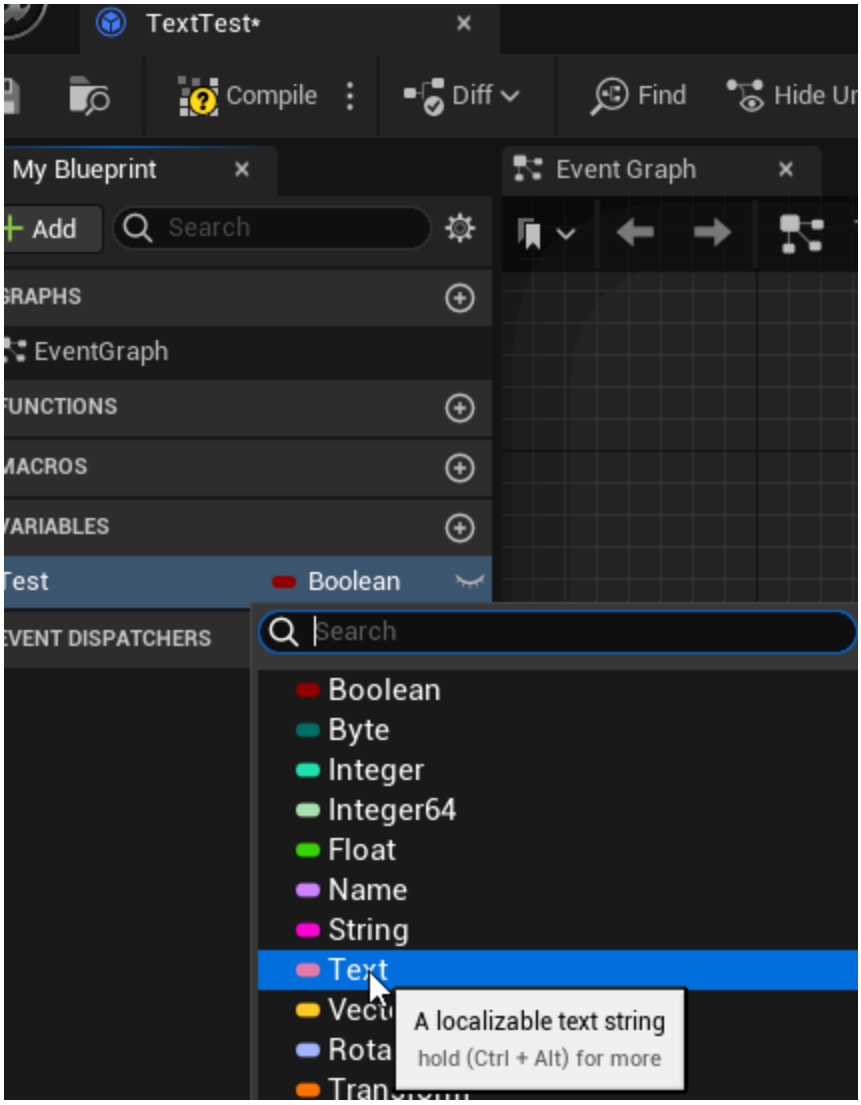
- Cut Text/Paste Text позволяет перенести текст из одного ассета в другой, сохранив при этом переводы и всю мета-информацию.
- Move to String Table позволяет перенести текст из ассета в строковую таблицу, заменив текст на ссылку.

Проверка работоспособности плагина

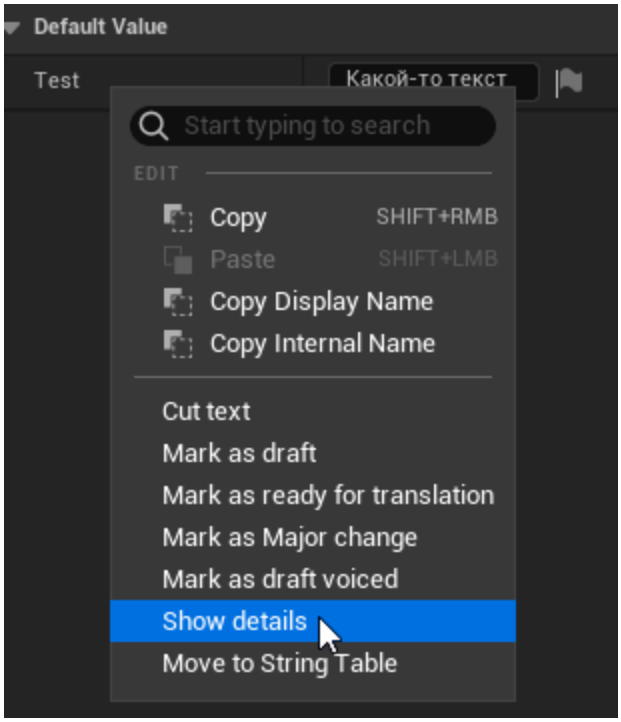
Действия совершаются в новом пустом проекте Unreal (Empty Project - C++) после установки и настройки плагина с зависимостями.

Создание текстов, редактор текстов

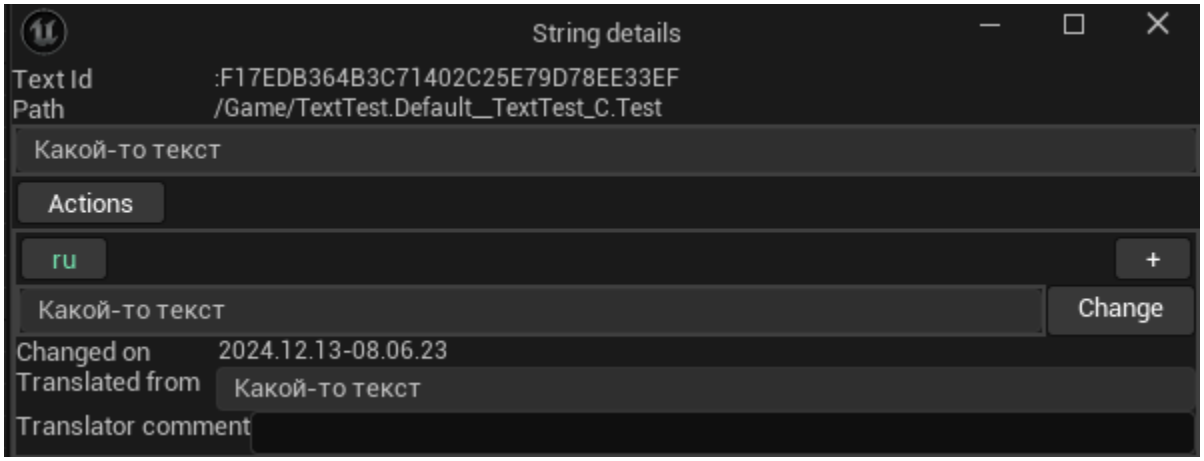
- Открыть в Content Browser папку Content (**не** какую-либо вложенную)
- Правый клик в окне Content Browser (в пустой части), выбрать Blueprint Class
- Выбрать Object в качестве базового класса
- Ввести имя тестового ассета TextTest
- Двойным кликом открыть редактор блюпринта. Добавить новую переменную Test с типом Text

- 

- Скомпилировать блюпринт, выбрать новую переменную. В панели Details в области Default Value ввести произвольный тестовый текст (например “какой-то текст”)
- Еще раз скомпилировать блюпринт
- Правый клик на имени переменной Test в окне Details должен показать следующее контекстное меню



- Выбрать в меню Show Details. Откроется окно информации о строке



- Проверить, что в строке есть текст и путь

Экспорт текстов для внешнего инструмента

- Открыть при помощи Проводника папку Content\Strings\Game в проекте (т.е. Если проект находится в C:\MyProject, полный путь будет C:\MyProject\Content\Strings\Game)
- Проверить наличие файла TextTest.F17EDB364B3C71402C25E79D78EE33EF.json (конкретный код после “TextTest.” будет отличаться - он должен совпадать с кодом в окне String Details)
- Открыть файл текстовым редактором (например Notepad)
- Удостовериться, что в файле корректный json с текстом и переводом. Файл должен выглядеть примерно так:

```
{
  "textId": {
    "Namespace": "",
    "Key": "F17EDB364B3C71402C25E79D78EE33EF"
  },
  "ownerPath": "/Game/TextTest.Default__TextTest_C.Test",
  "source": "ru",
  "source_text": "Какой-то текст",
  "source_modification_date": 638696739831910000,
  "comment": "",
  "speaker": "",
  "speaker_gender": "",
  "languages": [
    {
      "locale": "ru",
      "sourceLocale": "",
      "text": "Какой-то текст",
      "modification_date": 638696739831910000,
      "original_text": "Какой-то текст",
      "traits": []
    }
  ],
  "string_traits": []
}
```

Инструмент локализации. Инструкция

Инструмент локализации - приложение для просмотра и управления всеми локализованными текстами в игре. На данной странице описана его функциональность и детали реализации. Проектноспецифичные особенности работы указаны на соответствующих страничках в пространстве проектов.

Как понять, что программа работает?

1. Запустить Localization Tracker через ярлык
2. После небольшой паузы программа запускается. Открывается основное окно и слева над деревом папок, пока идет сканирование, появляется лейбл «Сканирование…»
3. Через некоторое время лейбл пропадает и появляется дерево папок, которое полностью повторяет структуру папки со со строками, указанной в конфиге
4. При клике на любую папку в дереве, в основном окне появляется таблица со всеми строками, которые есть в этой папке
5. При двойном клике на строку ее можно будет отредактировать, а при клике на другое поле, состояние строки сохранится
6. Через контекстное меню можно открыть файл строки и проверить внесенные изменения.

Введение

Основные понятия

В локализации и работе с инструментом необходимо понимать значение специфических терминов:

1. Строка - единица локализации, соответствующая отдельному файлу .json в репозитории, содержащая исходный текст, все тексты переводов и метаинформацию.
2. Локаль - язык, на котором написан текст строки. Одна строка может содержать текст в нескольких локалях.
3. Источник - тот язык, с которого осуществляется перевод
4. Цель - тот язык, на который осуществляется перевод
5. Трейт - пользовательское свойство, назначаемое на строку или локаль, используемое для фильтрации строк.
6. Тег - специальная метка, задающая форматирование текста или позволяющая отображать определенный текст вместо себя.

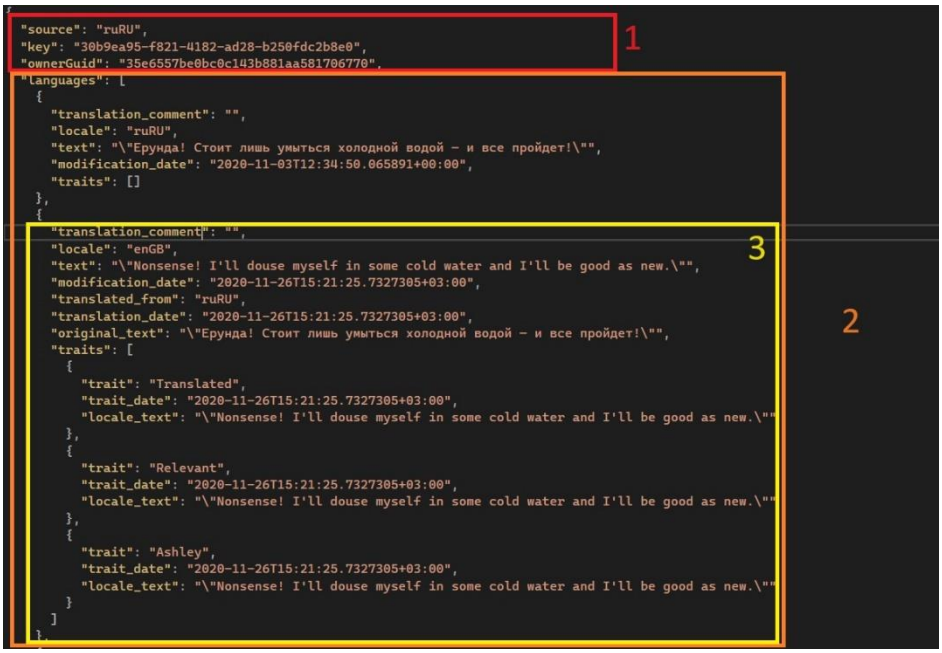
Обзор

Инструмент работает со строками, сохраненными в формате .json. Каждый файл содержит одну строку и все существующие переводы (локали) текста этой строки. Эти файлы ассоциированы с объектами в игровом движке. Таким образом, изменение строк в Инструменте или в игровом движке обновляет эти .json файлы. Таким же образом, если будут изменены строки в самих .json файлах, то все обновленные данные попадут и в Инструмент, и в движок.

Данные

Структура .json файла

Один .json файл хранит данные для одной строки по следующей структуре:



1.
    1. Блок, в котором хранятся данные для строки (номер 1):
        1. source - с какого языка переведено
        2. key - уникальный ключ стринги
        3. ownerGuid - ключ объекта на стороне движка, к которому эта стринга привязана
    2. Блок languages, содержащий все существующие для этой стринги локали (номер 2)
        1. Блок с информацией по каждой локали (номер 3)
            1. translation_comment - комментарий, который добавляется для подробного описания контекста ситуации, если это важно
            2. locale - язык локали

3. text - содержание локали, соответственно
4. modification_date - дата и время, когда последний раз было изменение текста в локали
5. translated_from - с какого языка был осуществлен перевод
6. translation_date - дата и время, когда был создан перевод
7. original text - соответственно, содержание локали, с которой был осуществлен перевод
8. traits - список трейтов, использующихся для локали

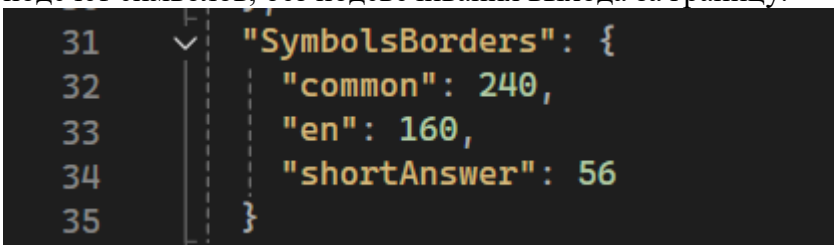
3. Блок string_traits, содержащий список трейтов для всей стринги.

Структура файла config.json

```
{
  "Project": "KristaLocTool",
  "IconPath": "Icon.png",
  "MultilineSearchIcon": "MultilineSearchIcon.png",
  "StringsFolder": "./TestProject/Content/Strings",
  "DialogsFolder": "./TestDialogs",
  "IgnoreMismatchedTags": [ "mf", "name", "n" ],
  "NeedClosingTags": [ "n", "g", "d", "m" ],
  "Locales": [ "en", "ru" ],
  "AddDefaultLocales": false,
  "AttachmentsPath": "./TestProject/Content/Localization/Attachments",
  "ShowStatusColumn": true,
  "DeepL": {
    "APIKey": "put-your-key-here",
    "SourceLocaleMap": {
      "ru": "ru",
      "en": "en"
    },
    "TargetLocaleMap": {
      "ru": "ru",
      "en": "en-US"
    },
    "MFTags": [ "mf", "mfs" ]
  },
  "Glossary": {
    "GlossaryGSheetId": "put-your-id-here",
    "GlossaryJSONPath": "Glossary",
    "GoogleCredentialPath": "Credential.json"
  },
  "SymbolsBorders": {
    "common": 240,
    "en": 160,
    "shortAnswer": 56
  }
}
```

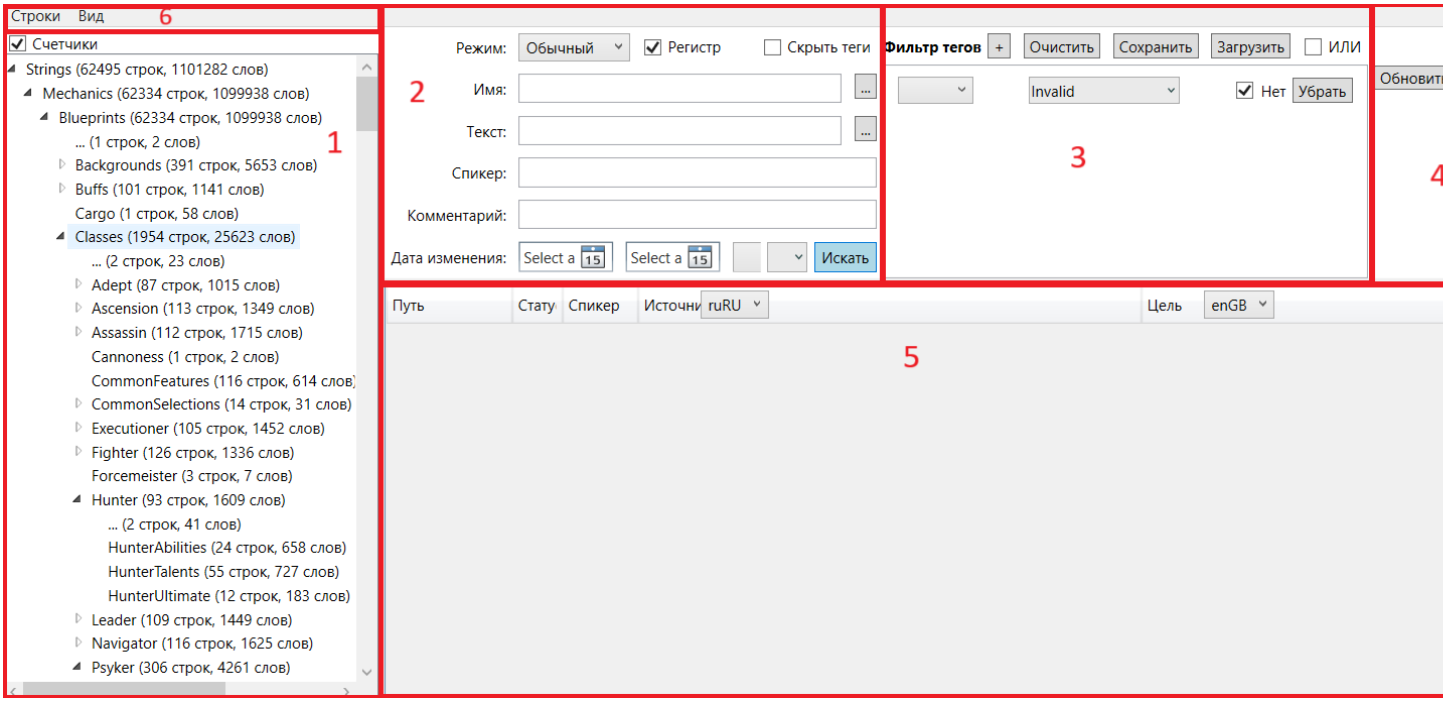
1. Project - здесь указывается название проекта. Оно высвечивается в названии окошка программы.
2. IconPath - это путь к иконке. Так как она лежит вместе с конфигом, то достаточно указать только название иконки.
3. StringsFolder, AssetsFolder, BlueprintsFolder - пути к соответствующим папкам. Инструмент может работать и только с папкой Strings, выдаст только предупреждение, что блюпринты и ассеты не определены.
4. IgnoreMismatchedTags - игнорировать указанные несовпадающие теги. Они не будут подсвечиваться, как несоответствие.

5. NeedClosingTags - теги, которые требуют закрывания.
6. Locales - поддерживаемые локали в дополнение к основным по умолчанию.
7. AddAIGeneratedTag - устанавливает, будет ли к переводам, выполненным через Deepl, добавлен блок [AIGenerated] в начале текста.
8. Область Deepl (выделена красным) - хранит данные, необходимые для корректной работы Deepl. API ключ, по которому получается доступ, а также языки для перевода.
9. ShowStatusColumn - в основном окне устанавливает видимость колонки Статус. Также если эта колонка не нужна - удалите поле или замените "true" на "false". Если на проекте в конфиге вы поля не видите, значит по умолчанию принимается значение "false"; чтобы это изменить, впишите поле самостоятельно.
10. Можно устанавливать границы на количество символов в строке, устанавливаются они также в конфиге. Если границы не установлены, то в окошке будет только подсчет символов, без подсвечивания выхода за границу.



Функционал

Основное окно



Основное окно включает в себя очень много различного функционала, поэтому будем рассматривать каждую часть отдельно.

1. Дерево директорий
2. Фильтры, поиск, режимы просмотра
3. Фильтры по трейтам
4. Кнопки с дополнительным функционалом
5. Таблица строк
6. Тулбар

Тулбар

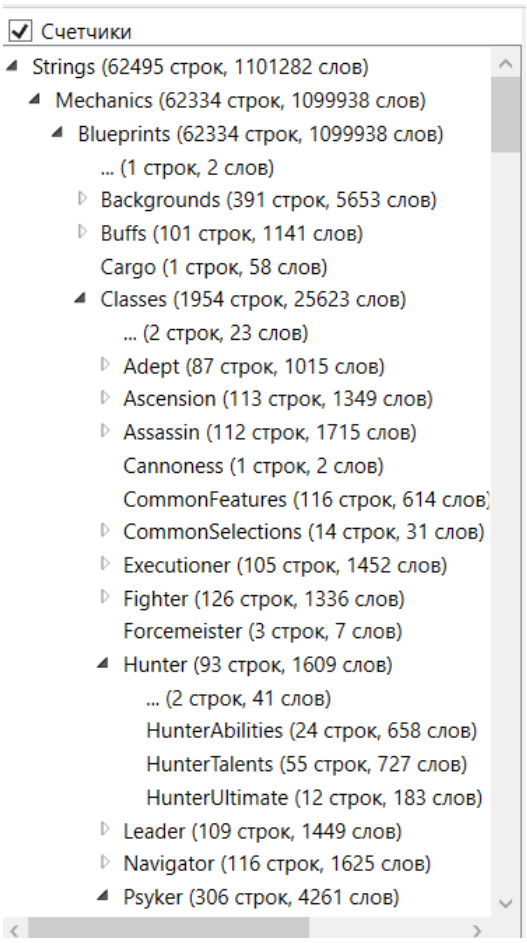
Строки

В разделе Строки по большей части повторение некоторого функционала, которое есть в основном окне. **Примечание: пока что не работают**

Вид

На данный момент работают 2 кнопки: Цвет фильтра и размер шрифта. Цвет фильтра повторяет функционал кнопки рядом с полем поиска Текст. Размер шрифта - позволяет устанавливать комфортный размер шрифта. Быстрее и удобнее работает на небольшом количестве строк, поэтому выбирайте для просмотра перед установкой не коренную папку, а поменьше.

Дерево директорий

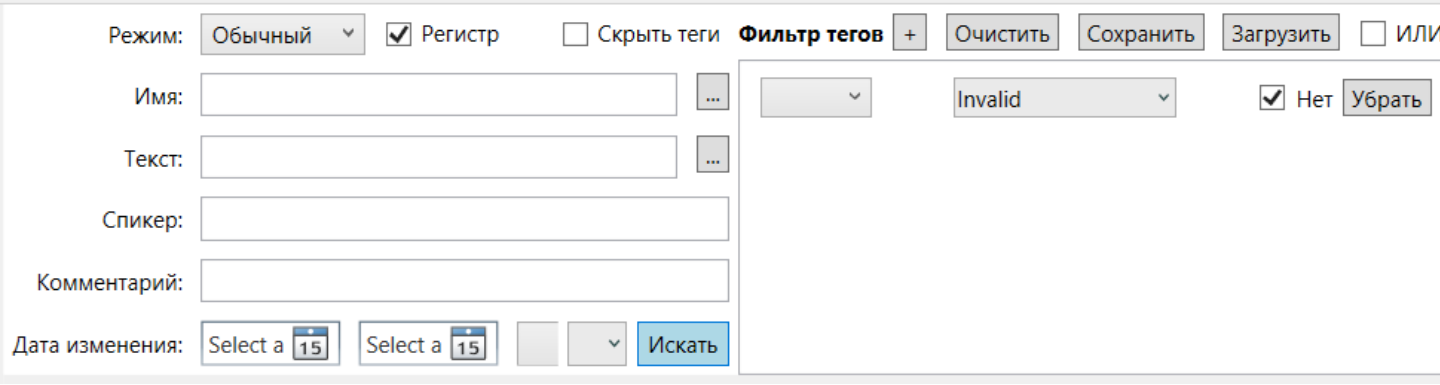


Активна галочка Счетчики, поэтому рядом с каждой директорией указано количество строк и слов

В отображении дерева папок есть несколько особенностей:
Вверху расположена галочка Счетчики, которая по умолчанию включена. Как следует из названия, можно увидеть количество строк и слов в каждой папке.

1. Директории можно сворачивать и разворачивать. Ваше личное состояние этих директорий (развернутое/свернутое) сохраняется при закрытии.
2. Чтобы в таблице строк появились строки, надо выбрать в дереве соответствующую папку, содержимое которой вы хотите увидеть.

Фильтры, поиск, режимы просмотра



Фильтры отфильтровывают строки соответственно тем полям, где вы вписали необходимое значение. В поле «Режим» необходимо выбрать режим, в котором вы хотите просматривать строки. Подробнее о режимах просмотра ниже.

Также в этом же окне есть чекбоксы, их назначение:

1. Регистр - игнорировать или не игнорировать регистр. По умолчанию всегда стоит галочка - игнорировать.
2. Скрыть теги - скрыть или отобразить теги. По умолчанию галочки нет - теги отображаются.

Как работает поиск:

1. После введения всех необходимых фильтров для поиска необходимо нажать клавишу Enter или Искать (голубая кнопка на скриншоте выше)
2. При полном удалении значения из строки поиск активируется автоматически

Поле Имя

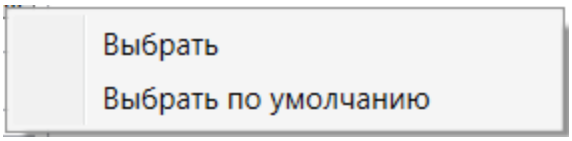
В этом поле поиск осуществляется по столбцу Путь. То есть по названиям файлов и директорий.

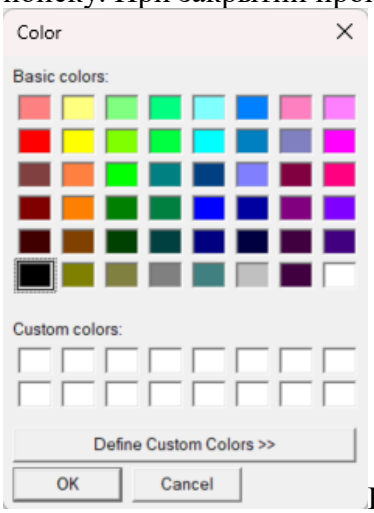
Справа от поля Имя есть кнопка с тремя точками, при нажатии на которую откроется окно для введения нескольких значений для поиска. То есть названия в пути файла можно фильтровать по нескольким значениям. После введения символов в это окно, в поле Имя появится строка <Multiple lines>. При закрытии окна множественного поиска значения внутри сохраняются.

Поле Текст

Здесь поиск осуществляется по полям Source или Target. При поиске по тексту в таблице строк выделяются совпадающие символы.

Справа от графы Текст также есть кнопка с тремя точками. При нажатии на правую кнопку мыши выпадет контекстное меню:



1. Выбрать - позволяет установить подходящий лично вам цвет выделения текста по поиску. При закрытии программы ваши изменения сохраняются.


Выбор цвета в стандартном меню, которое, думаю, почти всем знакомо
2. Выбрать по умолчанию - позволяет вернуть дефолтный цвет, сейчас он светло-синий, как на примере выше. за яркими

Цвет меняется в реальном времени, поэтому можете открыть нужный мод, нужные строки и проверять, какой цвет вам нравится больше.

Дата изменения

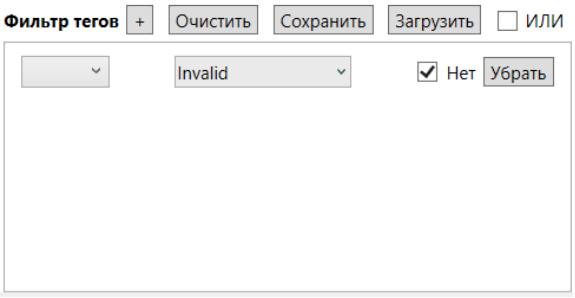
В поле «Дата изменения» можно выбрать локаль и трейты к ней, которые были изменены в какой-либо диапазон дат. Можно указать только дату крайнюю верхнюю, то есть модификации, начиная с выбранной даты , или крайнюю нижнюю - до какой даты.

Остальные поля поиска работают без каких-либо нюансов.

При использовании любого фильтра строки отфильтровываются в том числе в дереве директорий. Поэтому при введении фильтров отображаемые папки там будут меняться

Фильтры по трейтам

У фильтров по трейтам особый функционал



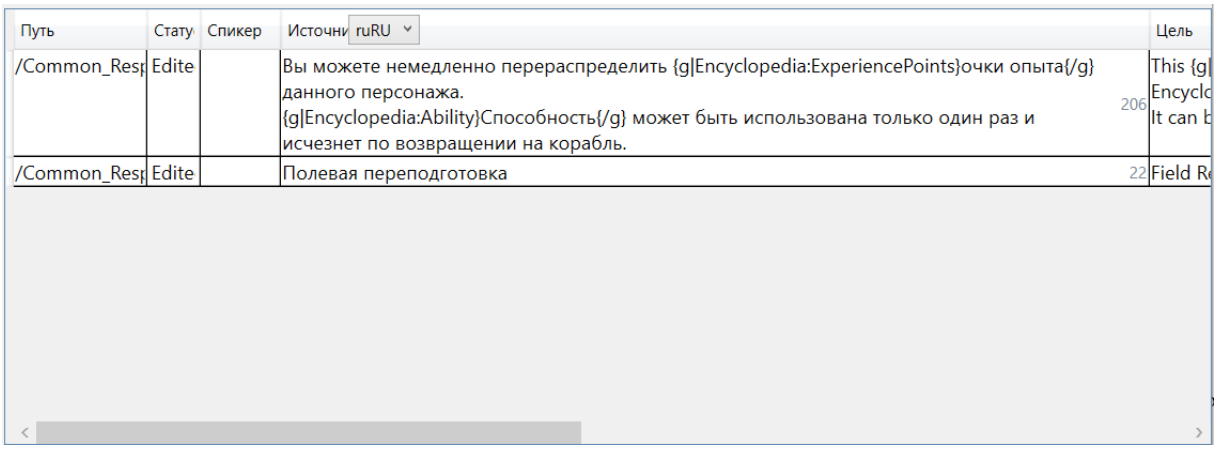
1. \+ - добавляет новый фильтр. В первом окошке при добавлении можно выбрать локаль (или пустую локаль), в следующем - интересующий трейт. Чтобы наоборот, вывести строки со всеми трейтами, кроме указанной, необходимо поставить галочку в чекбокс «Нет». Кнопка Убрать соответственно удаляет фильтр.
2. Очистить - позволяет удалить все внесенные фильтры
3. Сохранить / загрузить - сохраняет в .json файл / загружает из файла набор фильтров
4. ИЛИ- по умолчанию поиск всегда происходит через логическое "И", если проставить галочку, то поиск будет вестись через логическое "ИЛИ"

Кнопки с дополнительным функционалом

В правом верхнем углу находятся кнопки с дополнительным функционалом

1. Обновить словарь - обновляет термины в соответствии с глоссарием на гугл - диске.

Таблица строк



В моде Обычный таблица строк выглядит так. Красным выделены слова с неправильным написанием. Синим подчеркнуты термины, пояснение к которым можно найти в глоссарии в контекстном меню. В моде Обновлен_Трейт строки отображаются только по

локали-источнику, и необходимо выбрать не только язык, но и трейт, по которому ведется поиск. Во всех остальных модах отличается только отображение строк, состав таблицы не меняется.

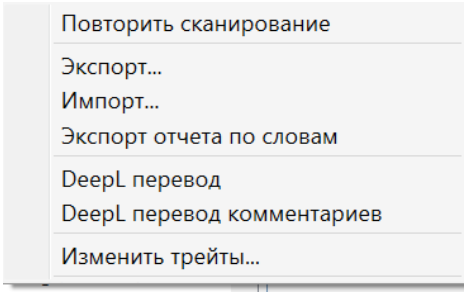
1. Путь - путь от выделенной директории в дереве директорий до файла строки.
2. Статус - показывает статус строки.
3. Спикер - кто говорит фразу.
4. Источник - строка с локалью, с которой перевели.
5. Цель - строка с локалью, на которую перевели.
6. Комментарий(источник)/Комментарий (цель) - комментарии к соответствующим локалям.

Режимы

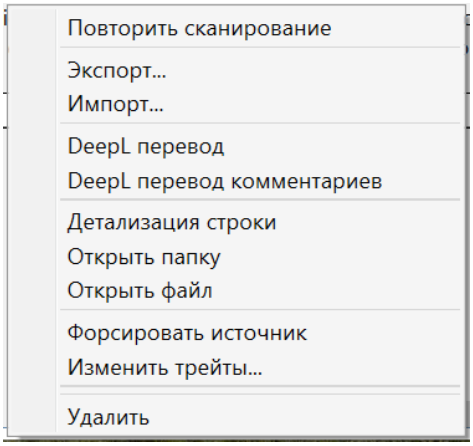
Строки можно просматривать в нескольких режимах (выбирается в выпадающем списке Mode):

1. Обычный - обычный
2. Проверка_Орфографии - отфильтровать только строки, в которых есть ошибки, выявленные спеллчеккером в какой-либо из двух выбранных локалей в колонках в списке строк.
3. Дубликаты_Ключей - показать только строки с неуникальными ключами. Дублирующиеся ключи - это ошибка в данных, которую нужно избегать всеми силами.
4. Обновлен_Исходник- показать только строки, в которых с момента перевода изменился текст в сорс-локали
    1. В правой колонке нужно выбрать локаль, на которую осуществлялся перевод
    2. В левой колонке нужно выбрать локаль Translation Source
    3. В этом режиме показываются диффы - что именно изменилось с момента перевода
5. Обновлен_Трейт - показать только строки, в которых в выбранной локали изменился текст со времени проставления определенного трейта
    1. В этом режиме в списке строк отображается только одна колонка с текстом
    2. В ней в заголовке нужно выбрать нужную локаль и трейт
        1. В этом режиме показываются диффы - что именно изменилось с момента проставления трейта
6. Разница_в_Тегах - показать несовпадающие в Источнике и Цели теги. Зеленым подсвечиваются теги, совпадающие в сорсе и таргете. Красным — есть в одной локали, но нет в другой (избыточные теги). И желтым - неполные парные теги.
7. Разница_в_Словаре - показывает только те строки, в которых термины в сорс локали и таргет локали не совпадает.

Контекстное меню



Контекстное меню для дерева директорий



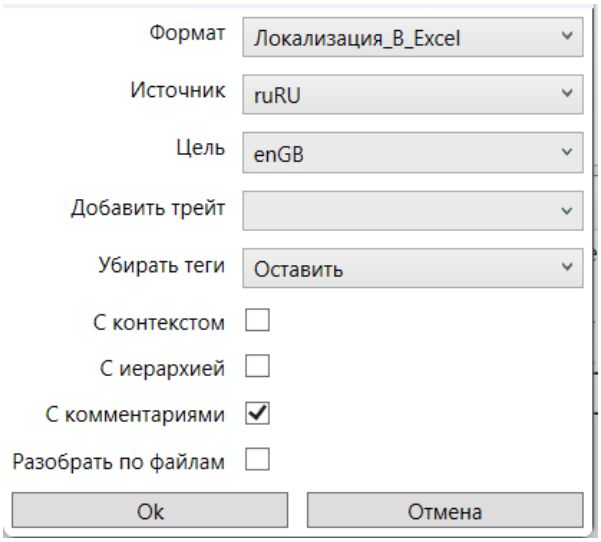
Контекстное меню для строк в таблице

Контекстное меню слегка различается для дерева папок и строк. Функционал из контекстного меню можно применить на нескольких выделенных строках, для этого строки необходимо выделить, зажав клавишу Shift.

1. Повторить сканирование - заново сканирует все строки, чтобы привести их в инстурменте в соответствие с текущим состоянием файлов .json в локальной копии репозитория.
2. Экспорт - экспортировать строки в Excel. Есть несколько форматов, об этом ниже
3. Импорт - импортировать Excel файл со строками. Программа сама автоматически их обновляет в соответствующих файлах. Важно: Excel с поддержкой макросов инструмент не увидит, поэтому перед импортом необходимо изменить формат таких файлов на обычный Excel.
4. Экспорт отчета по словам - экспортирует количество слов по всей структуре папок и подпапок выбранного диапазона. Экспортирует в structure.csv и сохраняет в папку Localization рабочей копии репозитория. Команда доступна только для дерева папок.
5. DeepL перевод - позволяет перевести с помощью DeepL строку. Если попытаетесь перевести таким образом строку, которая имеет не АИ-перевод, то выпадет предупреждение:
6. DeepL перевод комментариев - перевод с помощью DeepL комментариев к локалям. Комментарии подкладываются в соответствующую локаль. То есть если DeepL перевел с русского на английский, то его перевод попадет в локаль enGB с указанием, что это АИ-перевод.

7. Детализация строки - открывает окно с подробной информацией о всей строке, в том числе с историей изменений трейтов. Рабоатет только на одну выделенную строку.
8. Форсированный источник - назначает локаль, выбранную на момент исполнения команды в Источнике, copc-локалью для локали, выбранной в Цели.
9. Изменить трейты - позволяет изменить трейт для строки.
10. Удалить - удалить строку

Экспорт



Формат

При нажатии в контекстном меню строки «Экспорт» появляется следующее окно, в котором необходимо выбрать, что и как вы хотите экспортировать.

1. Локализация_В_Excel
   Это обычный экспорт выбранных строк. В этом режиме можно также экспортировать на несколько таргетных языков либо в один файл, либо каждый язык в отдельный. По умолчанию экспорт осуществляется в один файл, если поставить активный флаг в поле Разобрать по файлам, то для каждой локали-цели при экспорте создается отдельный файл.
   Таблица в экселе состоит из следующих столбцов:
   1. key - ключ строки
   2. name - путь до строки
   3. speaker - спикер соответственно
   4. source - текст в той локали, с которой осуществлялся перевод
   5. current - уже существующий перевод в локали-цели
   6. result - сюда необходимо вписать новый вариант текста локали
   7. comment - если галочка «С комментариями» активна (по умолчанию всегда активна), то добавляется еще столбец с комментарием к строке

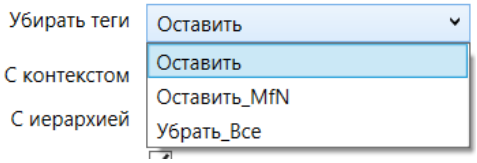
2. Различия_В_Excel - можно экспортировать строки с указанием удаленных символов и добавленных.
3. Разница_Тегов_В_Excel – можно экспортировать строки с выделением несовпадающих тегов.

4. Спикеры_И_Строки - экспортируется статистика строк и слов в copc- и таргет-локалях, а также количество строк с 'cue' и 'answer' в названии файла json, с разбивкой по спикерам в рамках выбранного диалога или нескольких диалогов.

	A	B	C	D	E	F	G	H	I	
1	Dialogues	Speaker	Strings	Words (in Strings) in SourceLocale [ruRU]	Words (in Strings) in TargetLocale [enGB]	Cues	Words (in Cues) in SourceLocale [ruRU]	Words (in Cues) in TargetLocale [enGB]	Answers	Words (in Answ
2	AbelardRecruit	None	18	143	199	0	0	0	18	
3	AbelardRecruit	AbelardCompanion	26	919	1186	26	919	1186	0	
4										

Убирать теги

Поле «Убирать теги» используется во всех форматах, кроме Спикеры_И_Строки.



1. Оставить - оставить все теги
2. Оставить_MfN - оставить только теги Mf и N
3. Убрать_Все - не оставлять никаких тегов

Чекбоксы

1. С контекстом - чекбокс добавляет контекстные строки. В xlsx файле в итоге перед каждой строкой будет "строка контекста" - предшествующий Cue или Answer (если есть).
2. С иерархией - экспорт с воссозданием иерархии папок и файлов для выбранных папок согласно иерархии в репозитории

Quest System

Плагин представляет собой систему для задания квестов для игры и изменения/отслеживания их статуса в рантайме. Плагин требует KristaMisc, KristaAssertions и GameRun для работы.
Для работы в рантайме требуется корректное включение GameRun в игре.

Как установить плагин в проект

1. Установить плагины KristaMisc, KristaAssertions и GameRun по их инструкции.
2. Перенести директорию QuestSystem в директорию Plugins проекта.
3. Запустить редактор
4. Перейти в меню Edit -> Plugins
5. Найти плагин QuestSystem и поставить галку напротив
6. Перезапустить редактор

Сущности в системе квестов

Quest Group. Группа - это категория в квест-логе. Группа квестов настаивается в отдельном ресурсе, она имеет текстовое поле "**Name**" - то, которое игрок будет видеть в квест-журнале. Также имеет поле Comments, которое никогда не видят игроки - это комментарии для внутреннего пользования. У квестовой группы есть **Приоритет** - цифра, которая указывает, на каком месте квест-группа расположена в квест-журнале.

Quest - "материнский" ресурс, содержит в себе список Objective-ресурсов и addendum-ресурсов. Имеет текстовые поля "**Name**", "**Description**". Также имеет поле Comments, которое никогда не видят игроки - это комментарии для внутреннего пользования.

Квест имеет свойство **Quest Group**, выпадающее поле где мы выбираем группу из всех имеющихся.

Галочки квеста: "**Main**" - если отметить эту галку, квест в квест-логе игрока будет помечен как важный, без выполнения которого невозможно продвижение по сюжету. Мейн-квест в логе всегда висит верхним в рамках своей подкатегории (QuestGroup).

Квест умеет находиться в следующих **статусах**:

None - никогда не выдавался.

Started - начат (выдан игроку). Не противоречит Failed и Completed - этот статус фиксирует факт выдачи, что мы этот квест когда-либо начинали, не важно, закончили или нет.

Active - активен сейчас, то есть Started, но не Completed и не Failed.

Failed - зафейлен.

Completed - успешно завершен.

Для квестов нет отдельных Action на выдачу, фейл и т.д. - квест автоматически выдается (становится Started) при выдаче любого обжектива квеста, а комплитится и фейлится по комплиту/фейлу обжектива, помеченного галкой **Finish Quest**.

Objective - ресурс, цепляемый к квесту. Имеет текстовые поля "**Name**", "**Description**", плюс поля "**Location(s)**" и "**Companion(s)**". Также имеет поле Comments, которое никогда не видят игроки - это комментарии для внутреннего пользования.

Галочки для обжективов:

Hidden - обозначает, что обжектив или аддендум не выводят в квест-лог игрока, но в остальном работают как обычные

Finish Quest - комплит этого обжектива комплитит весь квест, фейл этого обжектива фейлит весь квест.

Обжектив умеет находиться в следующих **статусах**:

None - никогда не выдавался.

Started - начат (выдан игроку). Не противоречит Failed и Completed - этот статус фиксирует факт выдачи, что мы этот обжектив когда-либо начинали, не важно, закончили или нет.

Active - активен сейчас, то есть Started, но не Completed и не Failed.

Failed - зафейлен.

Completed - успешно завершен.

Обжективы умеют выдаваться, комплититься и фейлиться **блюпринтовой функцией** - "**Set Objective Status**"

Если материнский квест становится Completed или Failed, все Active обжективы автоматически становятся Failed (логика такая - обжектив это отдельное задание, ты мог успешно закомплитить квест, не выполнив отдельные обжективы). После комплита квеста менять статус обжективов внутри него нельзя, они фризятся.

Objective-ресурс может содержать в себе список Addendum-ресурсов.

Addendum - ресурс, цепляемый к обжективу или квесту. Имеет только одно текстовое поле - "**Description**". Также имеет поле Comments, которое никогда не видят игроки - это комментарии для внутреннего пользования. Аддендумы - это небольшие дополнения и заметки.

Аддендум умеет находиться в следующих **статусах**:

None - никогда не выдавался.

Started - начат (выдан игроку).

Active - активен сейчас, то есть Started, но не Completed и не Failed.

Failed - зафейлен.

Completed - успешно завершен

Milestone - отдельный ресурс, имеет только поля "Name", "Description" и Comments. Может быть только **None** - никогда не выдавался и **Started** - начат (выдан игроку). Выдается отдельным экшеном **SetMilestoneSatus**.

Все сущности заводятся в контенте через контекстное меню.

Во время игры для работы с квестами используются функции из UQuestFunctionLibrary:

Для изменения состояния:

- `SetObjectiveState`
- `SetAddendumState`
- `SetMilestoneState`

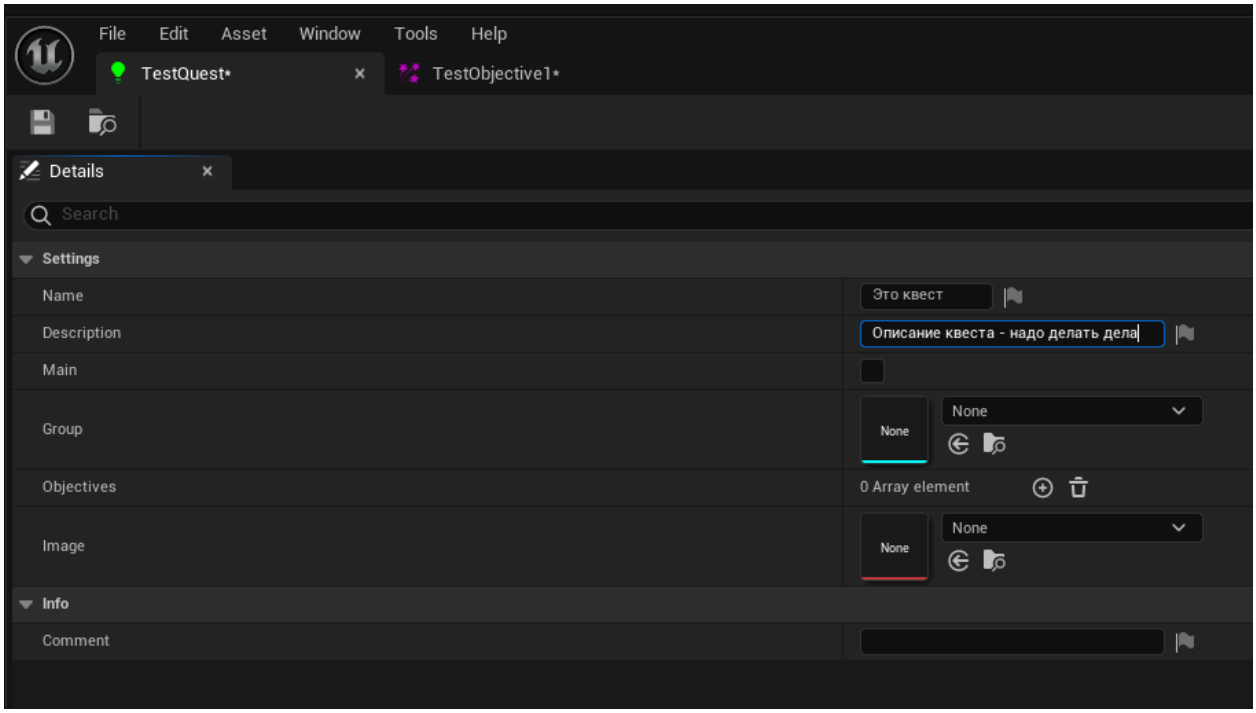
Для проверки состояния:

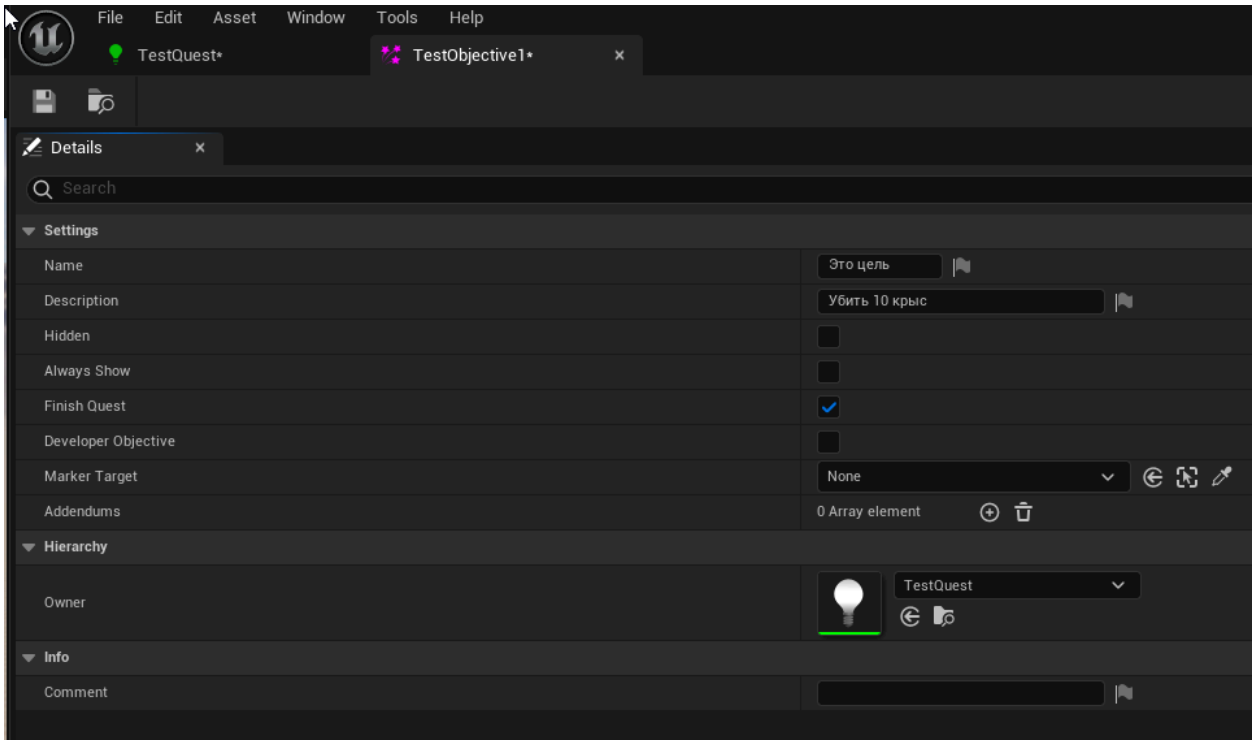
- `IsObjectiveState`
- `IsAddendumState`
- `IsQuestState`
- `IsMilestoneState`

Проверка работоспособности плагина

Действия совершаются в новом пустом проекте Unreal (Empty Project - C++) после установки плагина с зависимостями.

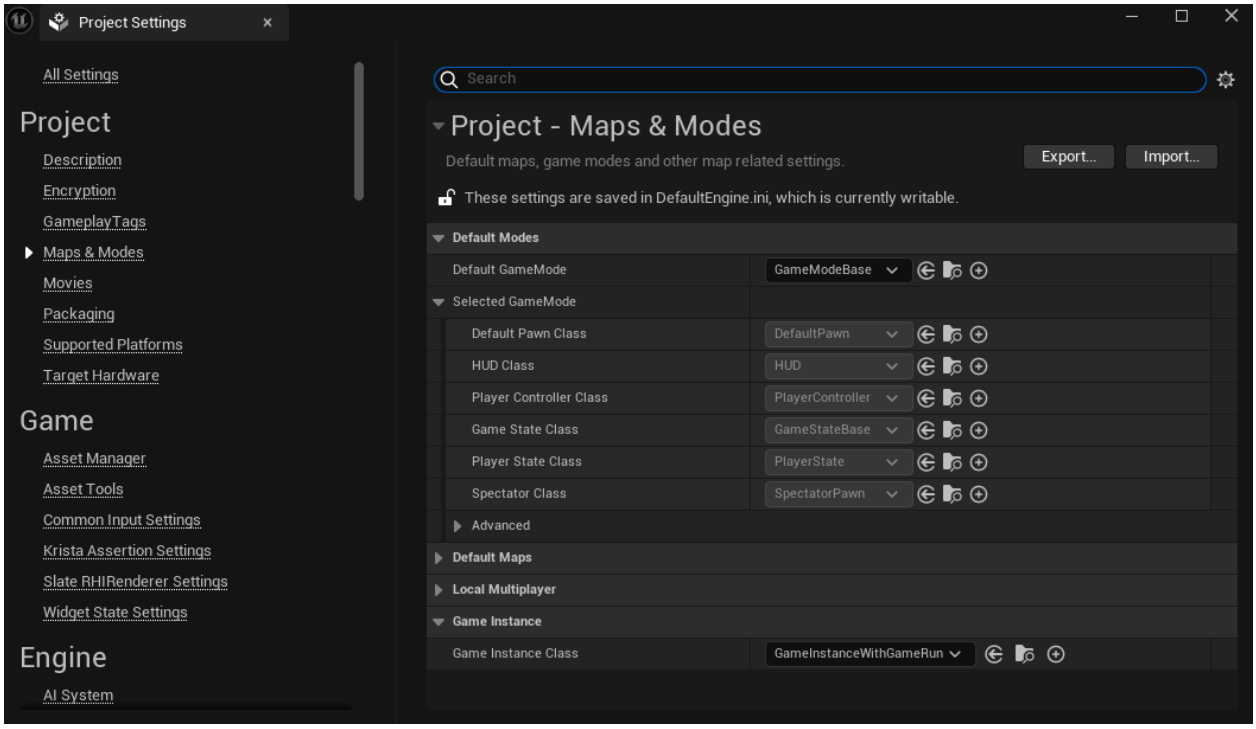
Создание квеста

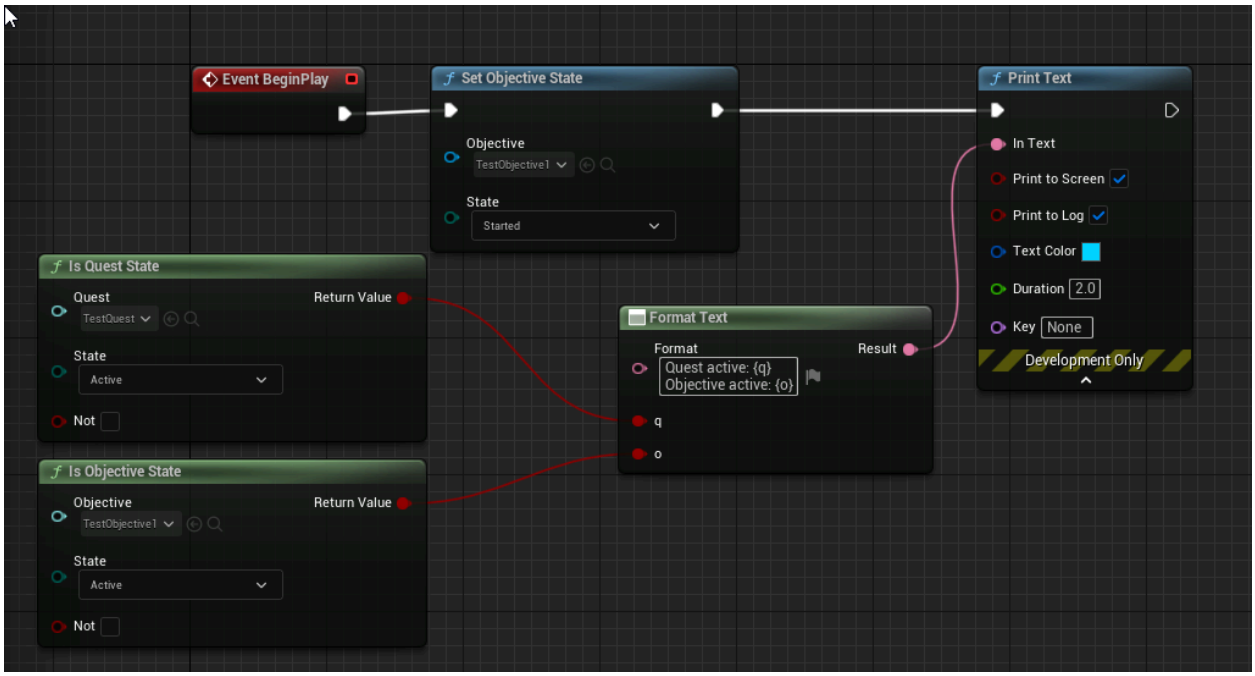
- Правый клик в окне Content Browser, выбрать Quest System/Quest
- Задать имя новому ассету (TestQuest)
- Двойным кликом по ассету открыть редактор. Ввести имя и описание квеста.
- 

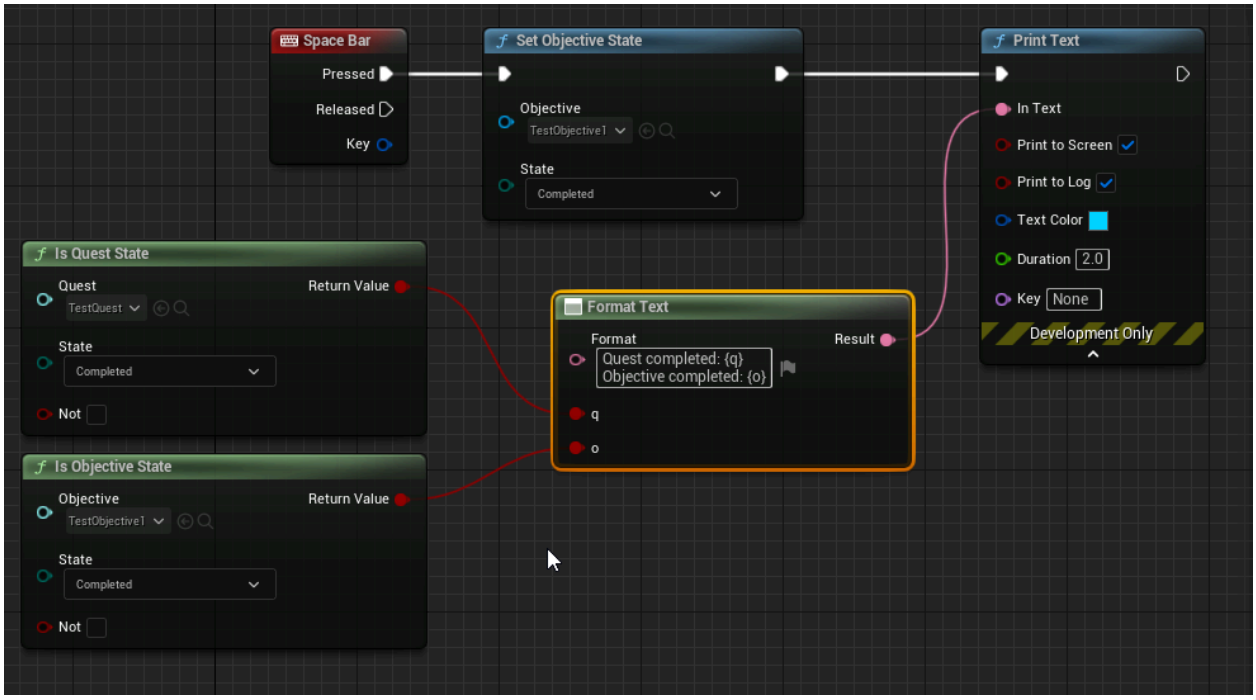
- Правый клик в окне Content Browser, выбрать Quest System/Objective
- Задать имя новому ассету (TestObjective1)
- Перетащить Objective в список Objectives в редакторе квеста
- Двойным кликом по ассету TestObjective1 открыть редактор. Ввести имя и описание обжектива, поставить галку FinishQuest
- 

Применение в игре

- Открыть меню Edit/Project Settings/Maps & Modes и выбрать в качестве GameInstance класс GameInstanceWithGameRun

- 

- Открыть блюпринт уровня и собрать следующий граф
- 



- Запустить игру на уровне, кликнуть на игровой области чтобы гарантировать положение фокуса ввода.
- Удостовериться, что на экране и в логе появилась надпись
  Quest active: true
  Objective active: true
- Нажать пробел
- Удостовериться, что на экране и в логе появилась надпись
  Quest completed: true
  Objective completed: true

Dialogue System

Плагин DialogueSystem предоставляет ресурсы, редактор и базовую реализацию системы проигрывания ветвящихся диалогов.
Плагин требует KristaMisc, KristaAssertions и GameRun для работы, а также LocalizationSystem, так как он используется для работы с текстом реплик. Для работы в рантайме требуется корректное включение GameRun в игре.

Устройство диалога

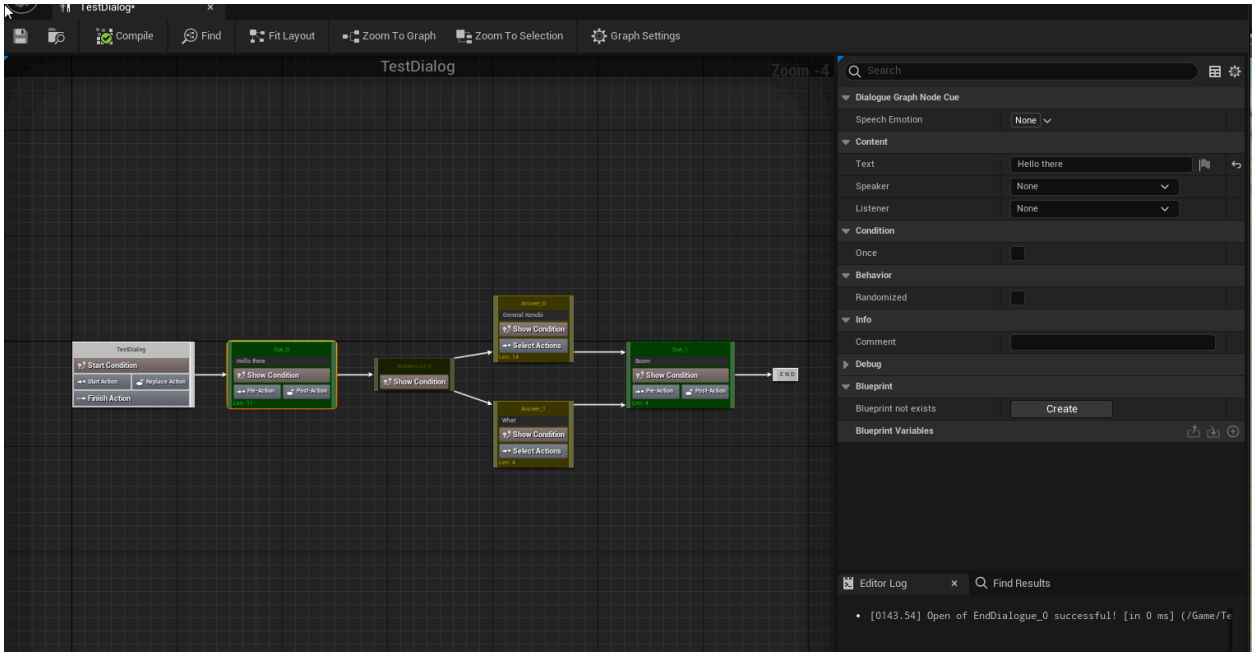
Диалог представляет собой граф узлов, представляющих собой отдельные реплики или списки реплик. Каждый узел это отдельный ассет в проекте, а диалог - папка с ассетами, входящими в общий граф. Поддерживаются следующие типы узлов:

- Root. Корневой узел, с которого начинается диалог
- Cue. Реплика игрового персонажа
- Answer. Реплика игрока - один из вариантов ответа на выбор
- AnswerList. Технический узел, предоставляющий возможность указать единый список ответов на какую-то реплику.
- CueSequence. Технический узел, предоставляющий возможность указать список реплик, проигрывающихся последовательно
- Action. “Пустая реплика,” узел, дающий возможность вставить комментарий или промежуток в диалоге, не сопровождающийся словами (например, для геймплейной сцены или катсцены)
- End Dialogue. Технический узел, маркирующий корректное завершение диалога.
- Goto Cue/Answer/etc. Технические узлы, позволяющие перейти на узел в другой ветви диалога.

Плеер диалогов в игре последовательно представляет игроку доступные узлы - реплики проигрываются в виде текста в интерфейсе (озвучивание голосом в плагине не предусмотрено, но соответствующий функционал несложно добавить в узлы самостоятельно). Варианты реплик игрока проигрываются в виде кнопок в интерфейсе, направляющих дальнее развитие по соответствующей ветке графа.
По умолчанию, при наличии нескольких доступных реплик, выбирается первая в порядке задания их в графе. Соответствующей настройкой можно поменять режим выбора на случайный.

Редактирование диалога

Новый диалог создается через контекстное меню Asset Browser. Двойной клик на любой ассет диалога открывает специализированный редактор

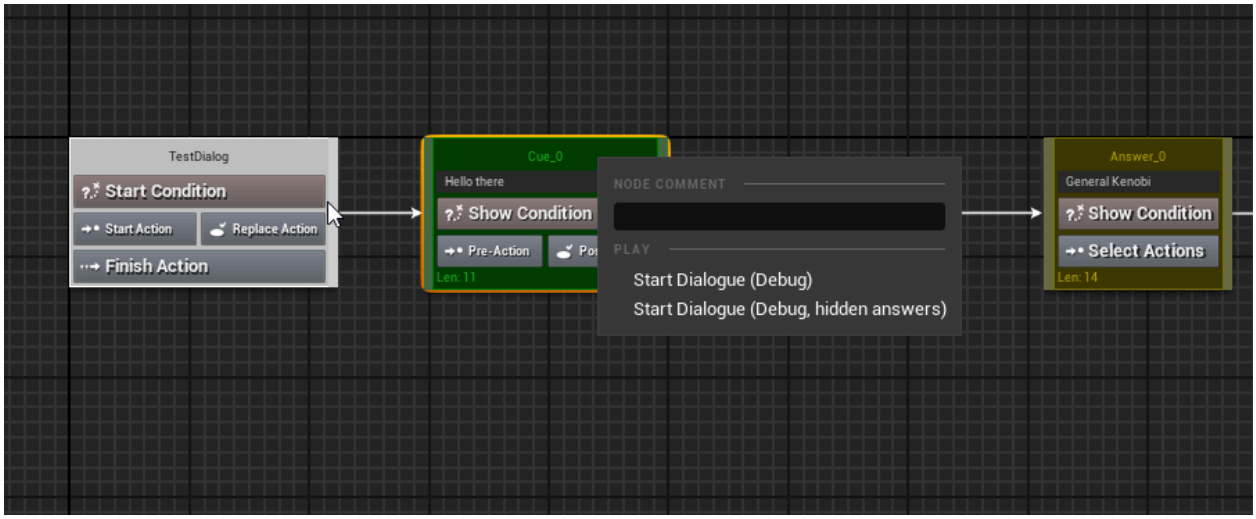


Редактор отображает диалог в виде графа, позволяет добавлять новые и удалять существующие узлы, задавать связи между ними и редактировать их свойства.
Новые узлы добавляются через контекстное меню, связи задаются через перетягивание из правой части узла.
Клик по узлу открывает его свойства в правой панели. Там можно задать комментарий разработчика, текст для реплик и говорящего его персонажа (в виде геймплейного тега).
Настройка Once для реплик отмечает узлы, которые будут показаны в игре только один раз. Настройка Randomized меняет способ выбора следующего узла (по умолчанию выбирается первый из возможных, с включением Randomized - случайный)
С каждым узлом может быть ассоциирован блюпринт. Это позволяет задать произвольную дополнительную логику для диалога: функции, которые вызываются перед/после проигрывания определенной реплики или при выборе определенного ответа, а также условия, которые проверяют доступность ответа/реплики во время игры.
Соответствующие редакторы для блюпринтов открываются по двойному клику на кнопки в узлах: Show Condition, Pre-action, Post-action, Select Action.

Использование в игре

Для использования диалогов в игре необходимо предоставить собственную реализацию класса `UDialogueGameSubsystem` (в минимальном виде можно не переопределять никакого поведения), а также проинициализировать `GameRun` - эта система используется для корректной инициализации и деинциализации диалогов.
Непосредственно интерфейс диалога в игре предоставляет реализация актора `ADialoguePlayer`. В плагине присутствует готовая реализация `ADebugDialoguePlayer`, которая реализует базовый функционал и пригодна для проверки работы.

Для запуска диалога используется функция `UDialogueGameSubsystem::PlayDialogue`. `UDialogueGameSubsystem::StartDebugDialogue` доступна в блюпринтах и позволяет проиграть диалог с использованием встроенного отладочного плеера. В режиме PlayInEditor из контекстного меню любой ноды диалога можно запустить отображение диалога с упрощенным UI.



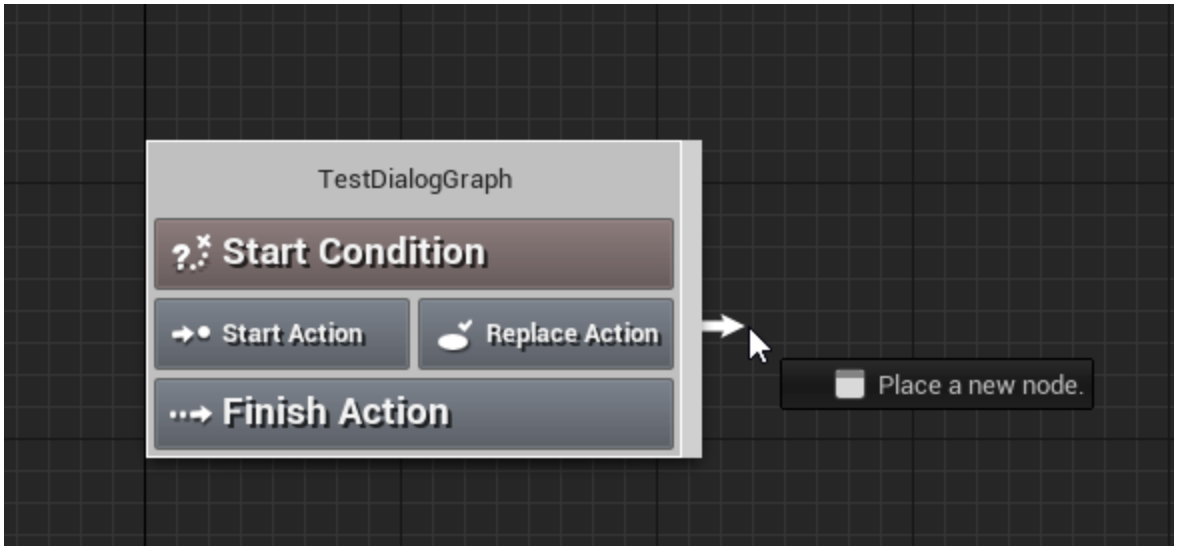
В библиотеке `UDialogueBlueprintFunctionLibrary` также определены функции, позволяющие проверять статус определенных частей диалога. Их можно использовать как в самих диалогах для задания логики, так и в любом другом геймплейном коде.

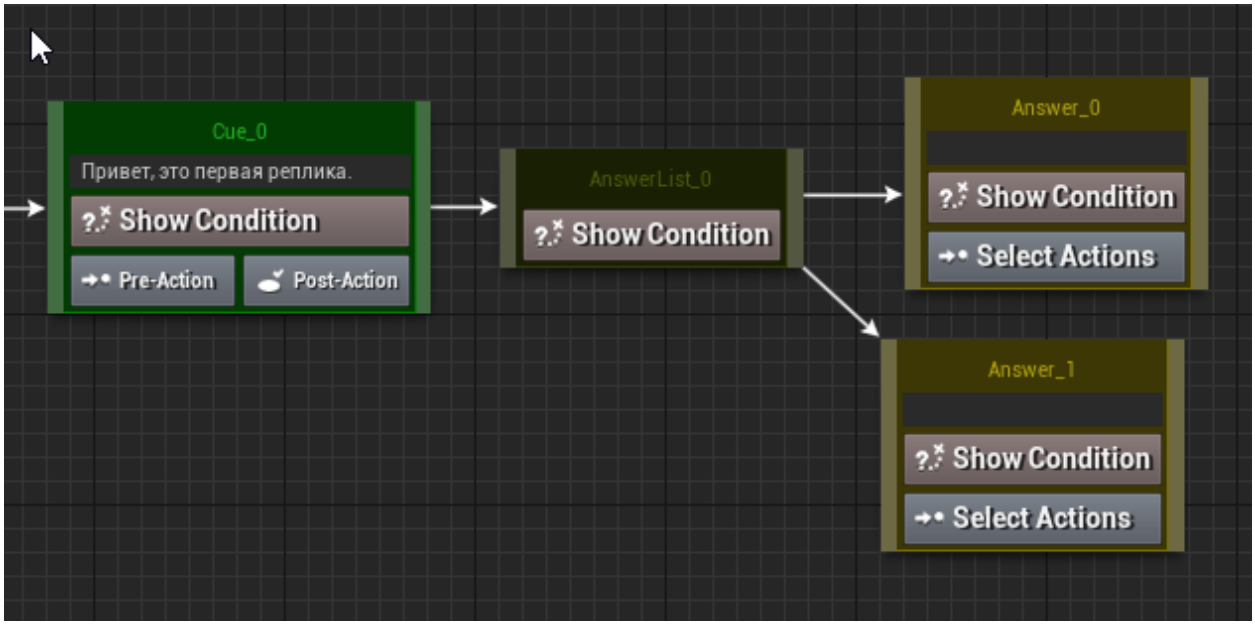
Проверка работоспособности плагина

Действия совершаются в новом пустом проекте Unreal (Empty Project - C++) после установки плагина с зависимостями.

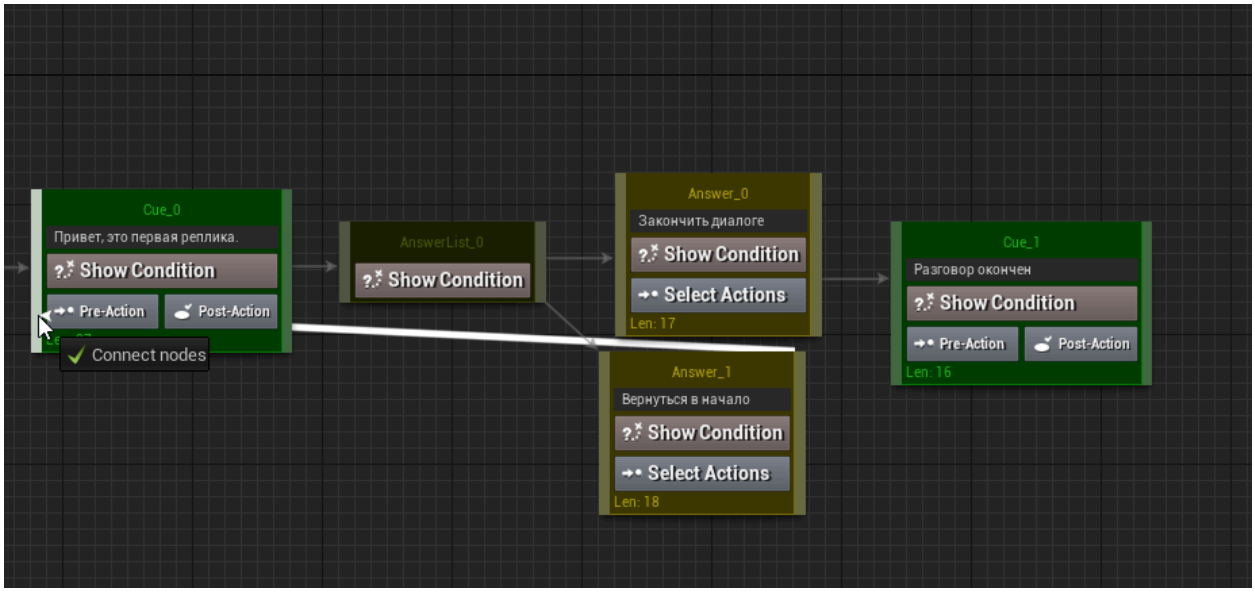
Создание диалога

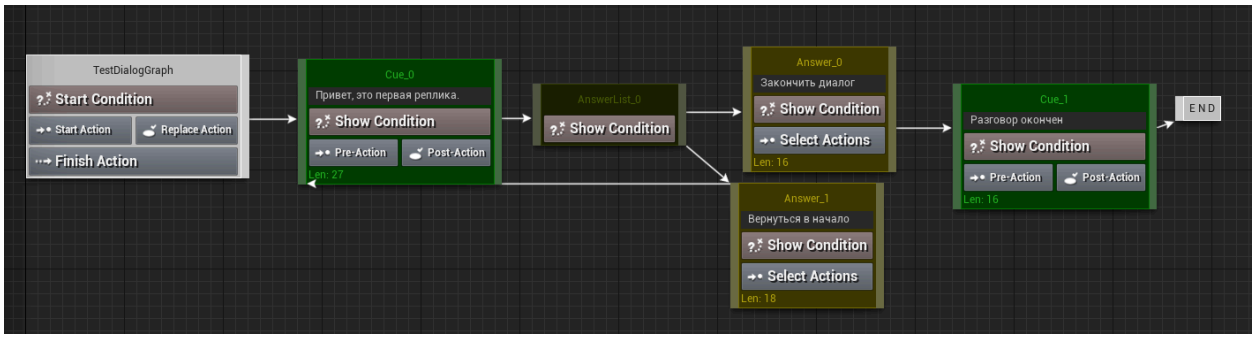
- Создать новую папку внутри Content с именем TestDialog
- Открыть её в Content Browser, в контекстном меню по правому клику выбрать Narrative/Dialogue
- Ввести имя нового ассета (TestDialogueGraph)
- Двойным кликом по ассету открыть редактор диалогов
- Зажать левую кнопку на полоске в правой части узла TestDialogueGraph и перетащить вправо. В открывшемся меню выбрать Cue

- 

- Кликнуть в поле ввода в появившемся узле реплики, ввести текст реплики (“Привет, это первая реплика.”)
- Аналогичным образом перетаскиванием из правой части реплики создать узел Answer List
- Затем перетаскиванием из правой части узла Answer List - два узла Answer
- 

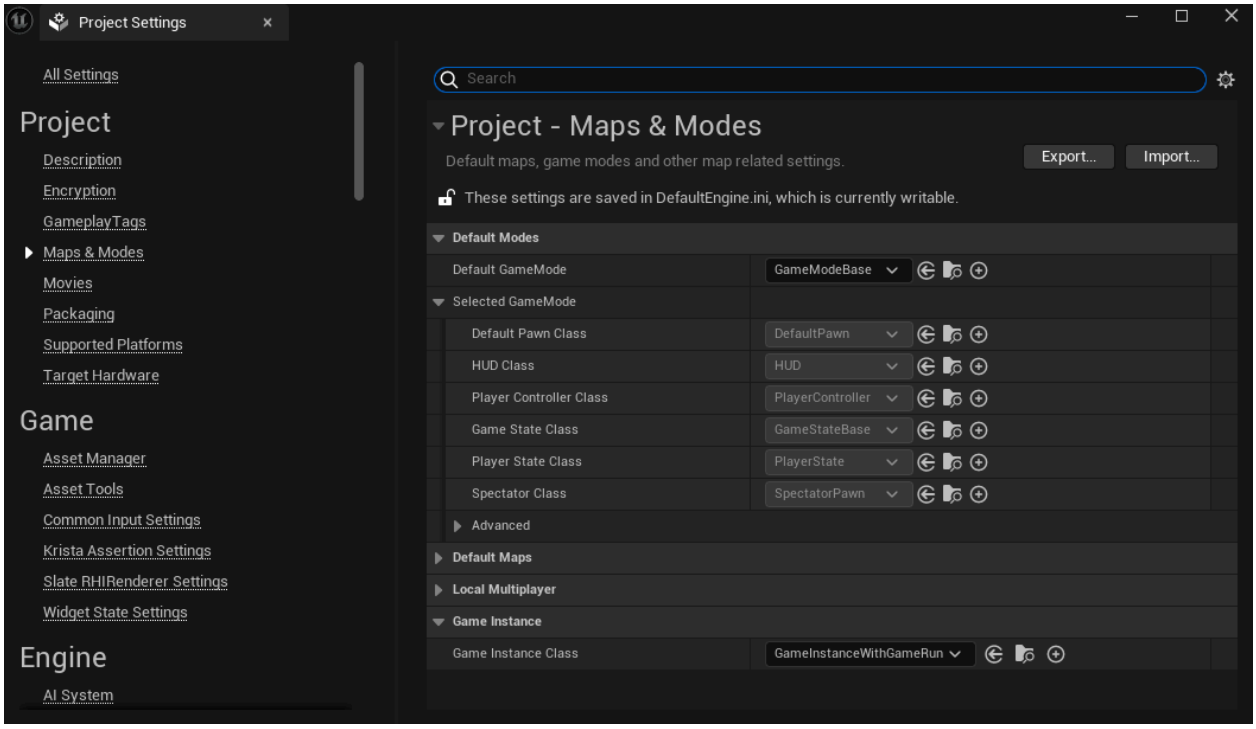
- Ввести тексты в узлы ответов: “Закончить диалог” и “Вернуться в начало”
- Перетаскиванием из первого Answer создать узел Cue. Ввести текст (“Разговор окончен”).
- Перетащить правую сторону второго Answer на полоску в левой части самой первой реплики (Cue_0)

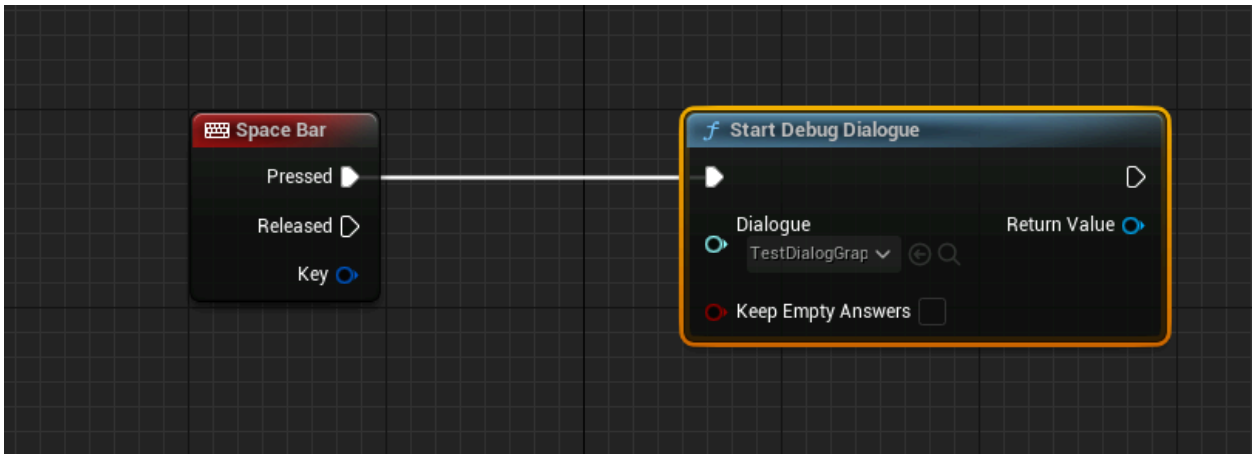
- 

- Перетаскиванием из правой части последней реплики (Cue_1) создать узел Dialogue End.
- Итоговый результат должен выглядеть так:
- 

Проверка диалога в игре

- Открыть меню Edit/Project Settings/Maps & Modes и выбрать в качестве GameInstance класс GameInstanceWithGameRun

- 

- Открыть блюпринт уровня и собрать следующий граф
- 

- В узле Start Debug Dialogue выбрать созданный выше TestDialogueGraph
- Запустить игру на уровне, кликнуть на игровой области чтобы гарантировать положение фокуса ввода.
- Нажать пробел
- Появится интерфейс диалога в игре
- 

- При клике на “Вернуться в начало” диалог возвращается к первой реплике (визуально ничего не меняется, т.к. она и так на экране)
- При клике на “Разговор окончен” появляется финальная реплика диалога

- Клик на Continue заканчивает диалог и убирает его интерфейс

Демонстрационный проект

В архиве DialogueSystemDemo лежит демонстрационный проект для плагина Dialogue System. Проект расчитан на Unreal Engine 5.4
Чтобы увидеть демонстрацию, нужно

1. Распаковать архив
2. Открыть DialogueSystemDemo.uproject редактором Unreal
3. Открыть карту TestMap и запустить игру
4. По нажатию пробела откроется диалог TestDialog, в котором будут видны реплики игры и возможность выбрать ответы игрока.

MisansceneSystem

Плагин MisansceneSystem предоставляет ресурсы, редактор и базовую реализацию системы последовательного и/или параллельного проигрывания специальных команд, в которых может быть реализована произвольная логика, специфичная для конкретной игры.
Плагин требует наличия плагинов KristaMisc и KristaAssertions для работы, а также обязательной реализации в своём проекте классов-наследников от AMisanscenePlayerBase и AMisansceneP oolBase(в минимальном виде можно не переопределять никакого поведения) и небольшую настройку.

Как установить плагин в проект

1. Установить плагины KristaMisc, KristaAssertions и GameRun по их инструкции.
2. Перенести директорию MisansceneSystem в директорию Plugins проекта.
3. Запустить редактор
4. Перейти в меню Edit -> Plugins. Найти плагин MisansceneSystem и поставить галку напротив. Закрыть редактор
5. Перейти в меню Edit -> Project Settings. Найти в секции MisansceneSystem пункт Misanscene Player Class и указать в нем класс вашей реализации AYourProjectMisanscenePlayer
6. Можно создавать собственные мизансцены и создавать акторы для проигрывания логики мизансцен на уровнях

Устройство мизансцены

Мизансцена представляет из себя граф узлов, позволяющих визуально запрограммировать логику исполнения команд. Каждая команда представляет собой отдельный ассет в проекте. Весь граф связей хранится в ассете самой мизансцены.
Граф мизансцены состоит из следующих типов узлов:

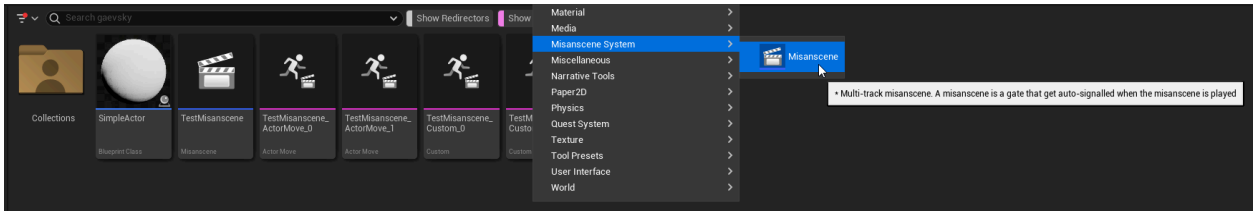
- Gate (в т.ч. Корневой узел мизансцены) - контейнер треков. Описывает порядок и логику запуска узлов типа Track
- Track - контейнер команд. Описывает порядок и логику запуска дочерних команд
- Command - непосредственные исполнительные элементы, реализующие необходимую логику.

Ассеты команд создаются исключительно в редакторе мизансцен. Удаление ассетов команд допустимо только из редактора мизансцен, который позволяет корректно разорвать ссылки между удаляемыми и оставшимися ассетами. По умолчанию ассеты команд скрыты в ContentBrowser. Для их отображения есть настройка Edit -> Project Settings -> Misanscene Editor -> Show Command Assets и фильтр для ContetnBrowser, который может управлять их видимостью для пользователя.

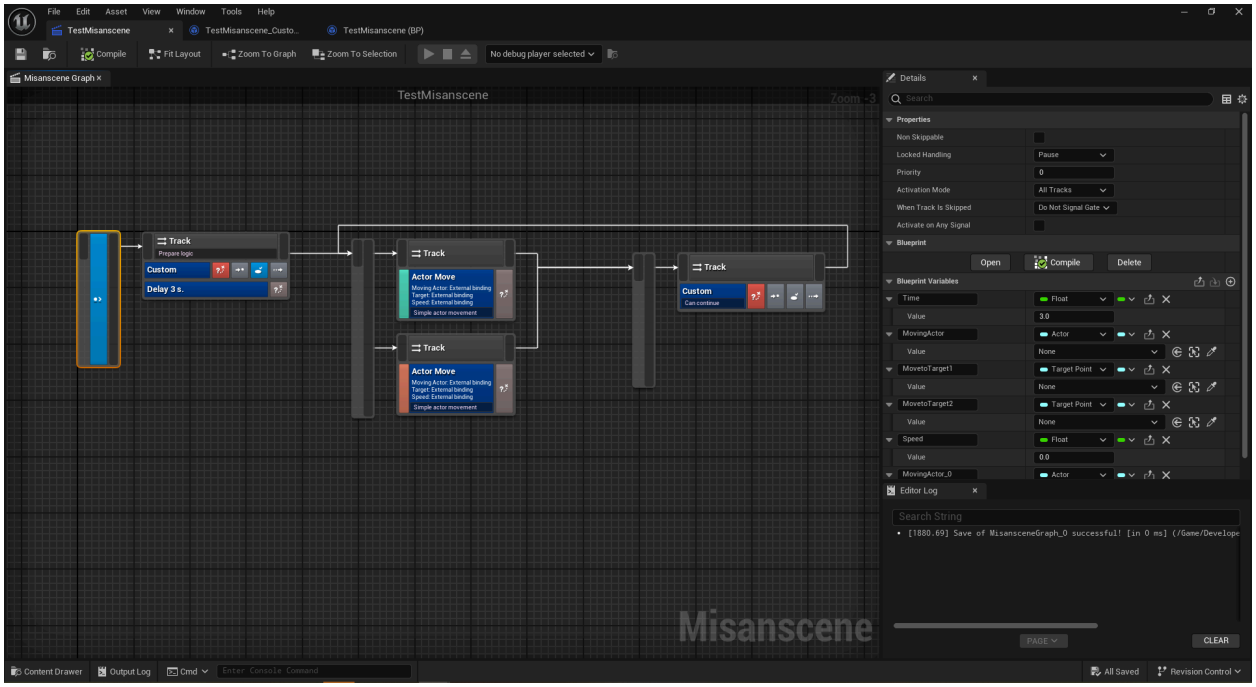
Для реализации команд со специфичной для конкретного проекта логикой необходимо создать реализацию дочерних классов от UMisansceneCommand и/или UMisansceneCommandTaskBase. Все классы команд, реализованные к конкретном проекте, автоматически становятся доступны в редакторе мизансцен.
В плагине поставляется команда UMisansceneCustomCommand, позволяющая всю необходимую логику реализовать в блюпринтах. А также команда UMisansceneDelayCommand, которая предоставляет функционал простой временной задержки.

Редактирование мизансцены

Новая мизансцена создается через контекстное меню Asset Browser. Команды можно создавать только из редактора мизансцен.Удаление команд также допустимо только из редактора мизансцен, который позволяет корректно разорвать ссылки между удаляемыми и оставшимися ассетами

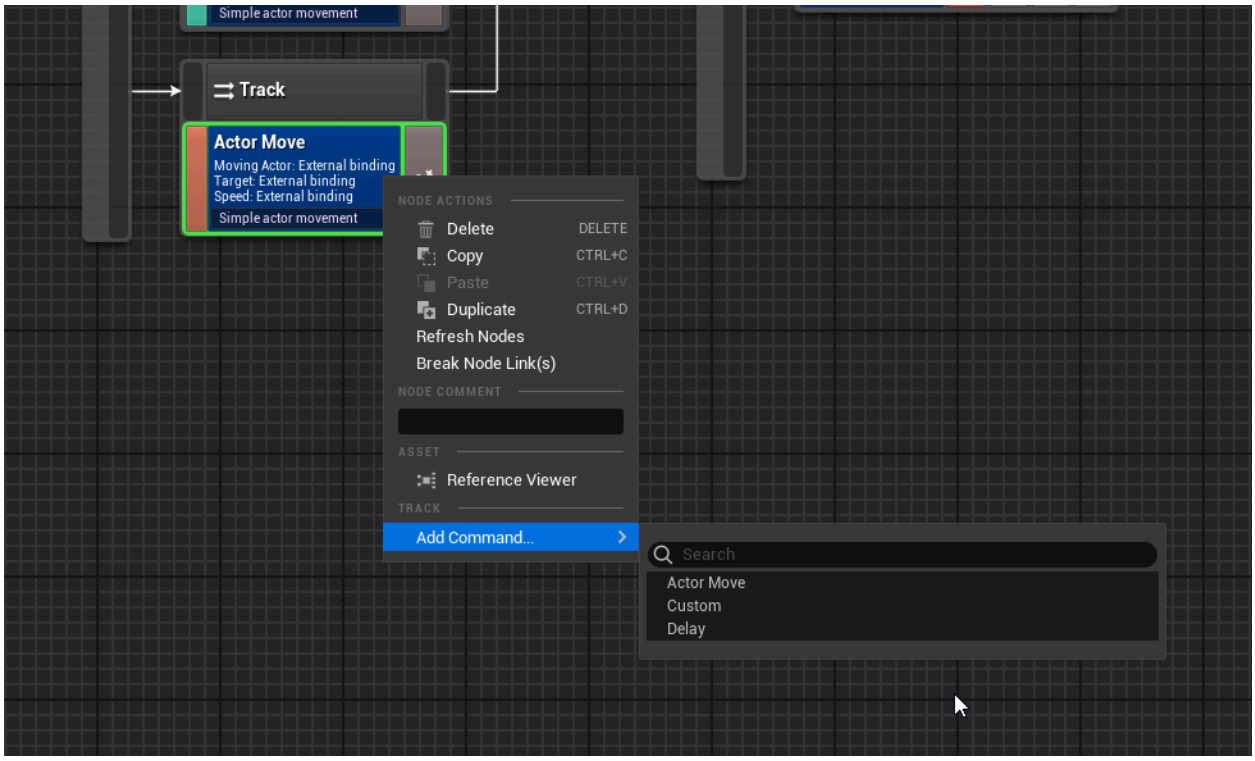


Двойной клик на любой ассет мизансцены или команды открывает специализированный редактор

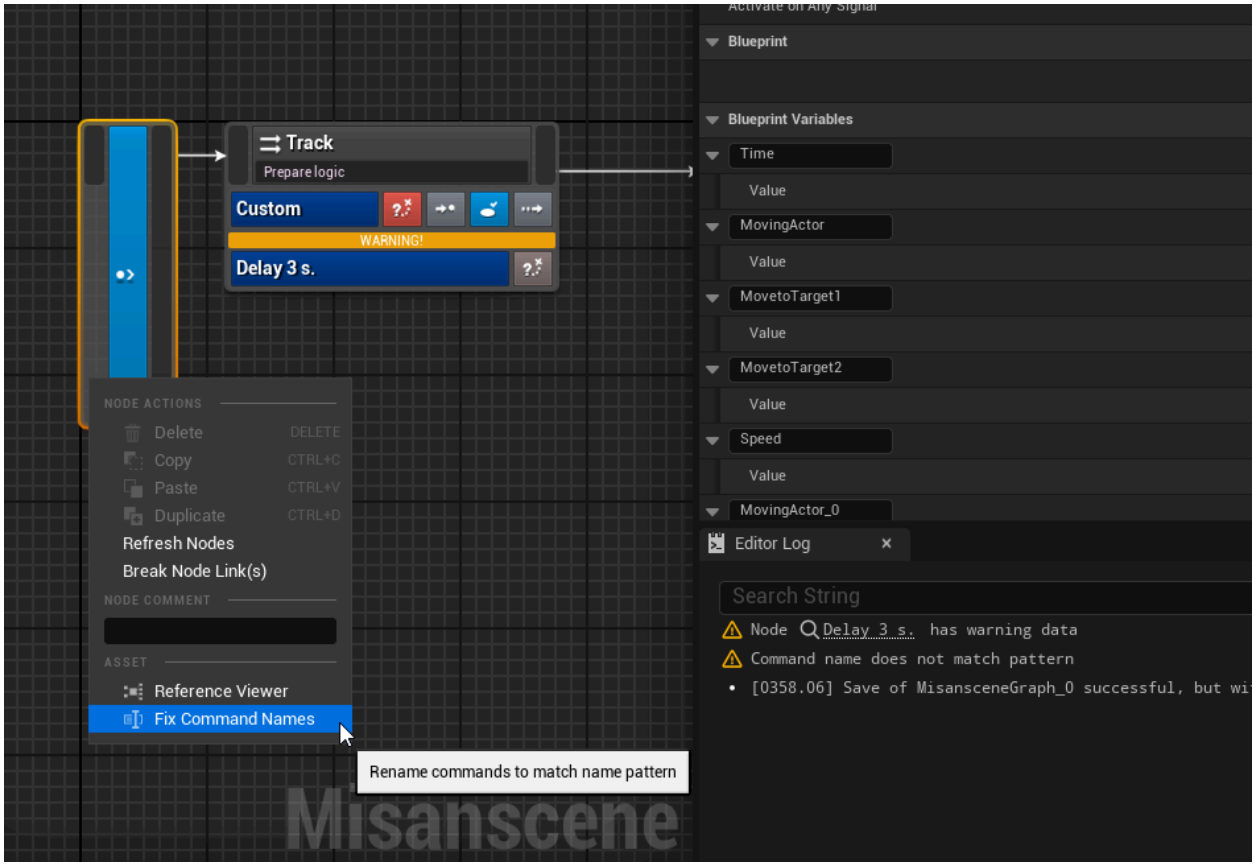


Вновь созданная мизансцена уже содержит корневой элемент.
Редактор позволяет:

- Редактировать свойства каждой ноды в панели Details
- Поддерживает механизм Undo|Redo
- Создавать новые ноды Gate и Track, а также дочерние ноды команд в нодах Track через контекстное меню.

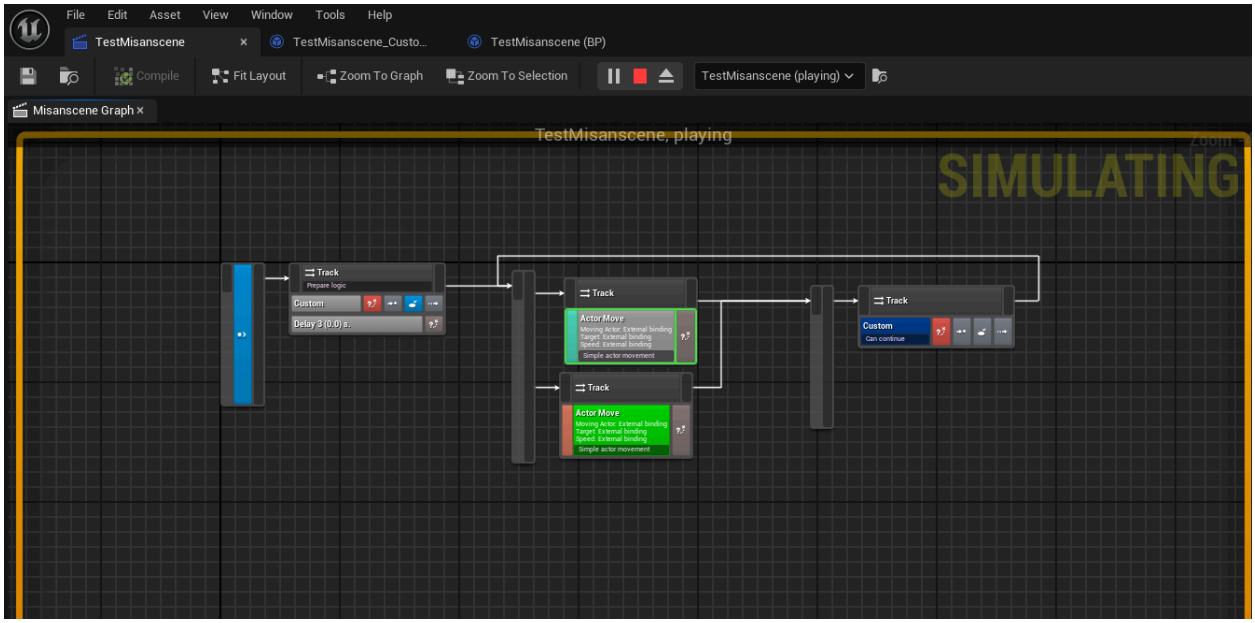


- При сохранении мизансцены запускается механизм валидации данных. Обнаруженные ошибки выводятся в лог, ошибочные ноды подсвечиваются
- Имена ассетов команд автоматически создаются по шаблону “ИмяМизансцены_*ТипКоманды*_Номер”. Новые ассеты команд создаются в папке родительской мизансцены.При переименовании ассета мизансцены валидатор команд будет определять их как не соответствующие шаблону. Автоматически переименовать дочерние команды в соответствии с шаблоном можно через контекстное меню корневой ноды мизансцены

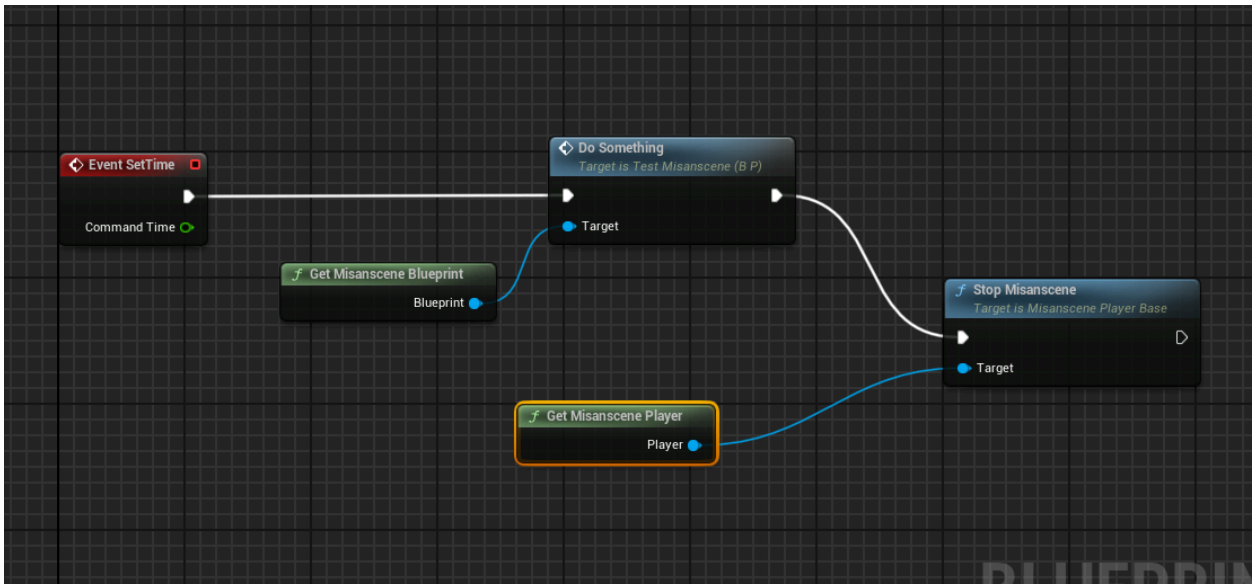


- Соединять произвольные треки с произвольными гейтами используя входные или выходные Pin-s соответствующих нод
- С помощью Drag’n’Drop менять порядок треков в гейте и команд в треке, переносить команды между треками
- Произвольно располагать гейты с дочерними треками на графе
- Добавлять комментарии к трекам и командам
- Копировать или дублировать выделенные ноды из редактора любой мизансцены в буфер обмена и вставлять с сохранением связей и иерархии. При этом создаются все необходимые ассеты
- Удалять выделенные ноды с проверкой конфликтов. Операция необратима.
- В режиме Play In Editor позволяет запускать, ставить на паузу, останавливать проигрывание мизансцены в выбранном MisanscenPlayer на загруженой локации. Выбирать управляемый MisanscenePlayer.Управление проигрыванием

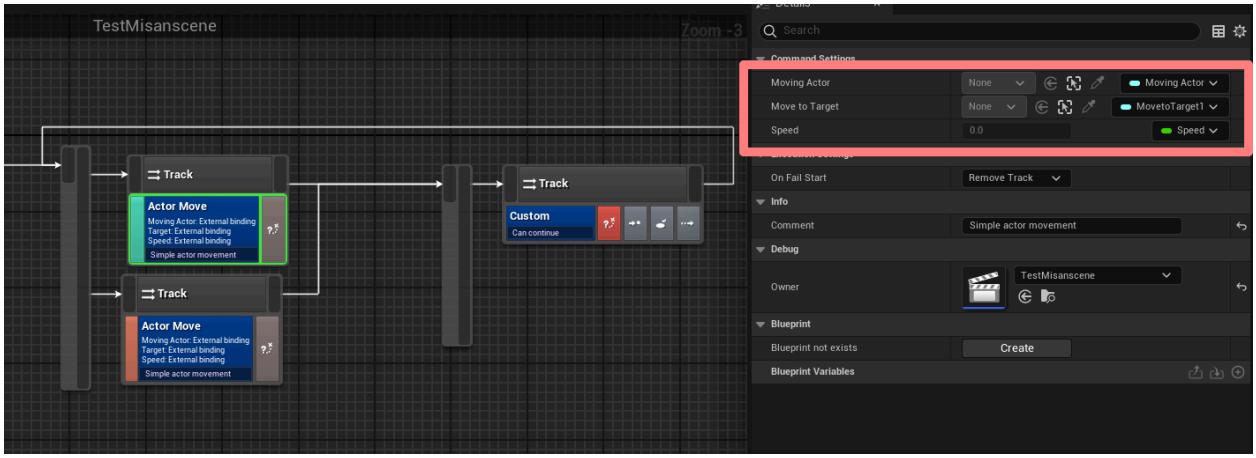
осуществляется специальным виджетом на панели инструментов



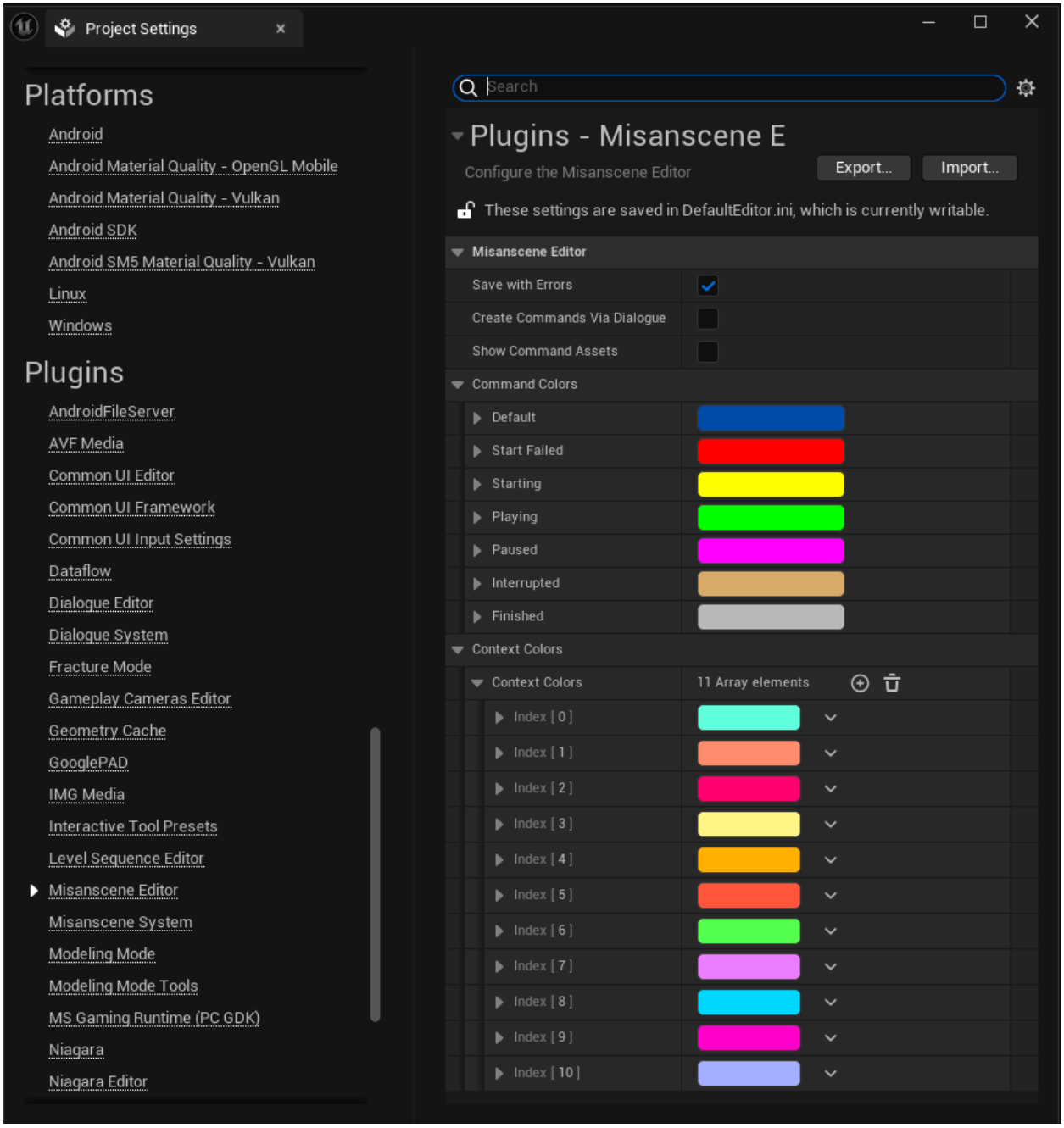
- Создавать дочерние скрытые блюпринты для самой мизансцены и для каждой команды в отдельности для реализации необходимой логики. Из блюпринта любой команды есть доступ к блюпринту родительской мизансцены или к управлению MisanscenePlayer-ом через специальные блюпринтовые ноды.



- Использовать механизм биндинга свойства команд на блюпринтовые переменные мизансцены. Эти переменные могут быть инициализированы нужными данными непосредственно в окне свойств плеера мизансцены на уровне



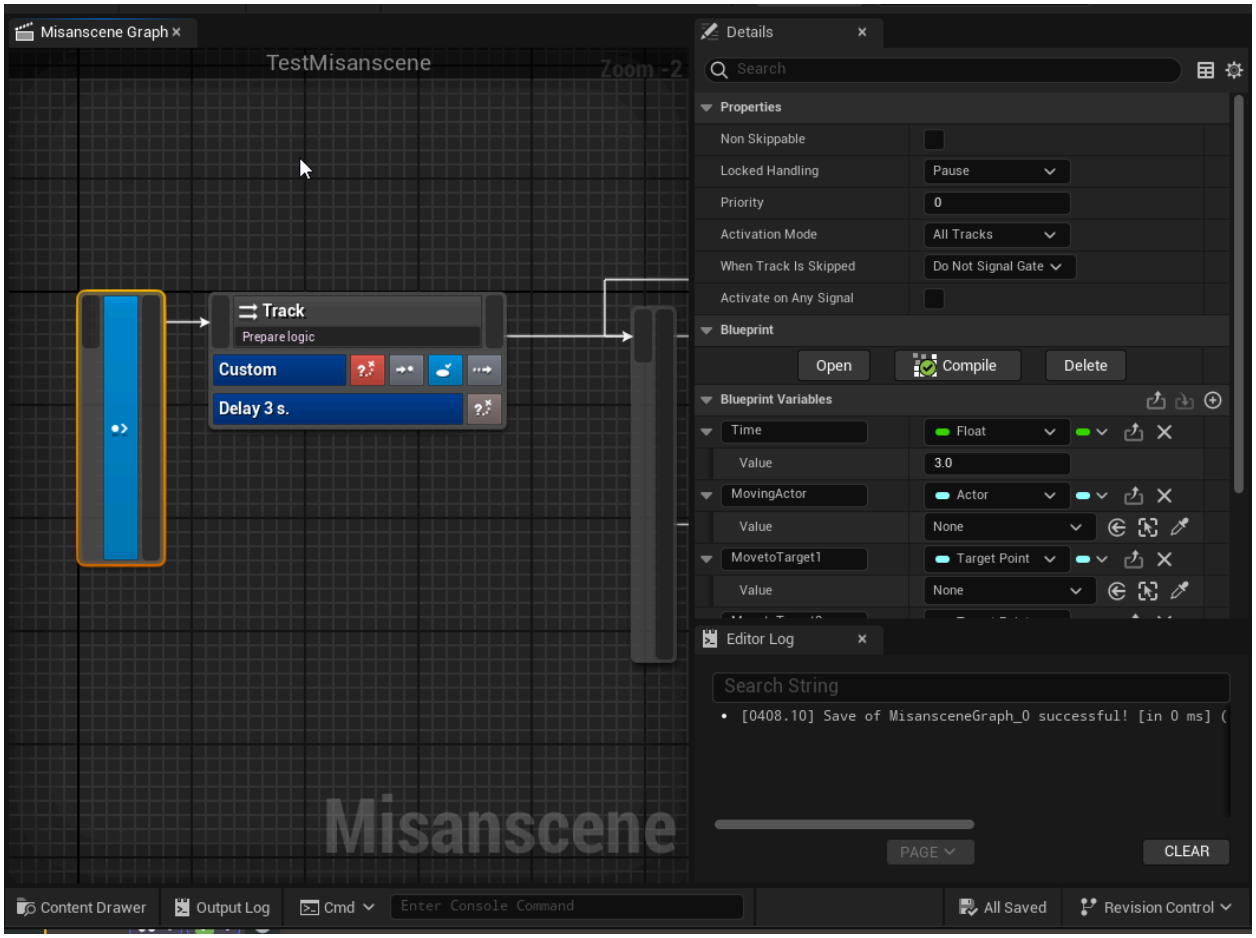
- Если в проекте определены собственные классы команд, которые используют предоставляемый функционал “контролируемого объекта” и контекста использования этого объекта (например, разные игровые персонажи имеют уникальный идентификатор, позволяющий их различать на локации), то такие команды получают дополнительную цветовую маркировку, соответствующую контексту. Уникальность контекста целиком лежит на стороне реализации пользовательской команды.
- Редактор имеет ряд настроек, который пользователи могут менять. Например цвета нод в различных состояниях, или цвета маркеров контекста



Описание узлов

Root

Стартовый Gate-узел, который хранит настройки логики работы всей мизансцены



Имеет настройки:

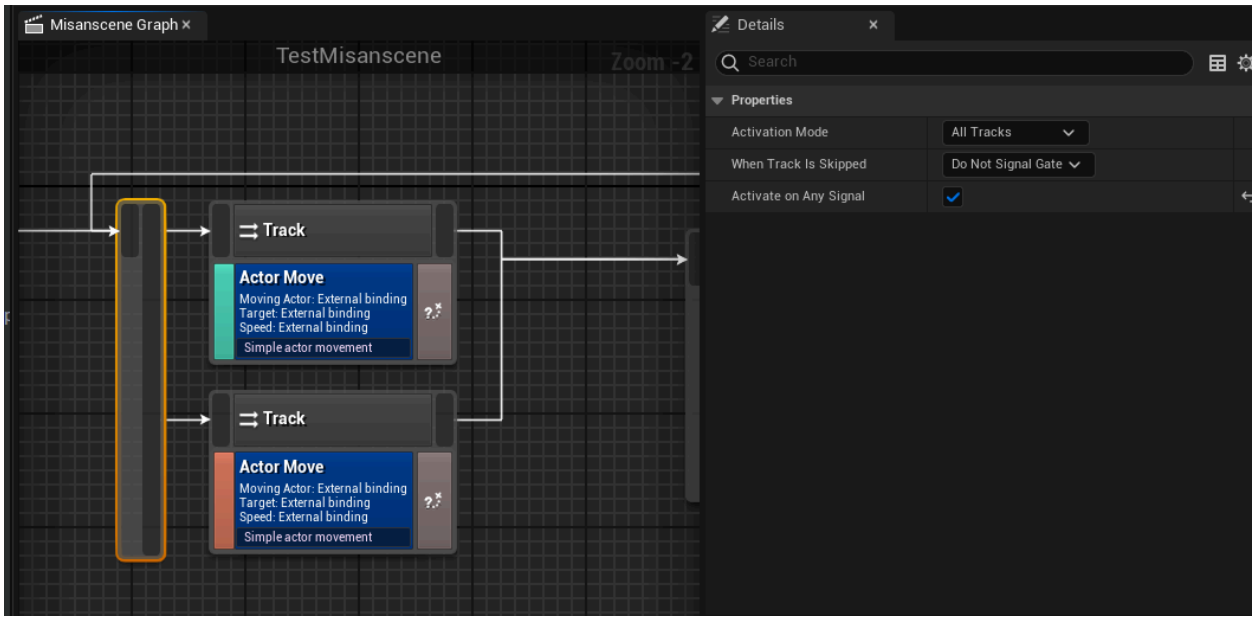
- NonSkipable - режим проигрывания мизансцены, когда командам запрещен пропуск логики выполнения. Режим пропуска логики должен поддерживаться пользовательскими командами. Например, персонаж в процессе перемещения после включения режима пропуска логики сразу телепортируется в конечную точку.
- LockedHandling - настройка поведения при проигрывании мизансцены когда несколько MisanscenPlayer-ов пытаются получить контроль над одним объектом, например управлять одним персонажем (требуется поддержка в реализации конкретных команд на основе единого механизма регистрации таких объектов в MisansceneSystem и их глобальной блокировкой внешней игровой логикой). Мизансцена может иметь произвольное число команд с конкурентным доступом к различным объектам. Если хотя бы один объект заблокирован другой мизансценой или внешней игровой логикой, то поведение мизансцены определяется исходя из значения этого свойства. Может принимать значения Pause, Stop, Pause and Restart
- Priority - приоритет мизансцены, который определяет поведение при блокирующем доступе к одному объекту
- На самой ноде есть интерактивная кнопка, которая позволяет автоматически создавать/открывать при двойном клике функцию, которая вызывается в момент завершения проигрывания мизансцены

- Набор виждетов управления дочерним блюпринтом. Виджеты в панели Details позволяют Создавать/Открывать/Удалять блюпринт, создавать, копировать, вставлять новые переменные

Остальные настройки идентичны ноде Gate

Gate

Гейт является контейнером активируемых им треков в соответствии с принятыми сигналами активации

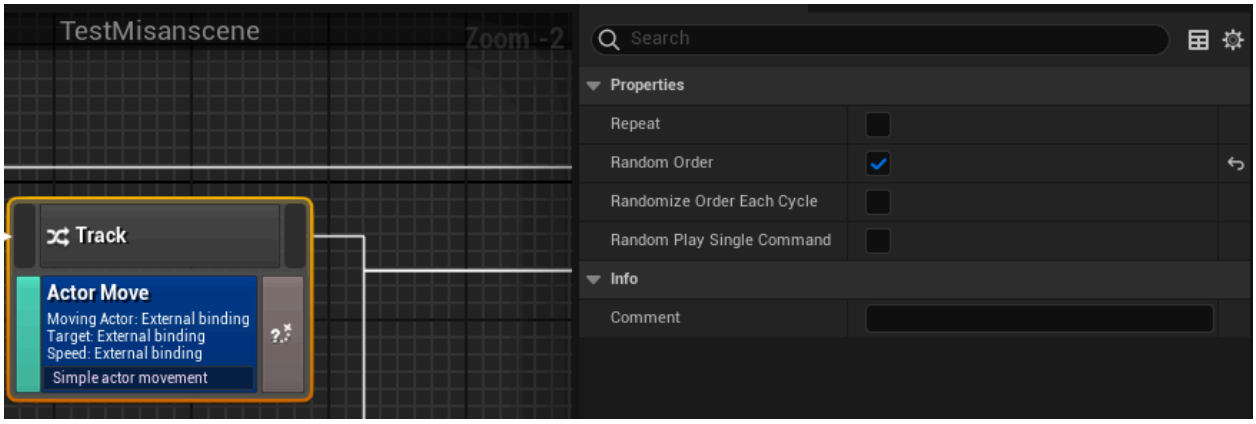


- ActivationMode - режим активации гейта, который определяет, какие треки будут запущены. Имеет значения All Tracks, First Track, Random Track. Режим отличный от All Tracks помечается информативным маркером над нодой. Порядок обхода треков при запуске - сверху вниз
- WhenTracksIsSkipped - задает поведение дочерних треков при завершении работы гейта. DoNotSignalGate - дочерние треки не посылают сигналов о завершении работы, SignalGate - посылают сигнал, но только те треки, которые не являются Continuous, т.е. Работу которых нельзя прервать (см. Описание ноды Track)
- ActivateOnAnySignal - гейт активируется сразу при получении входящего сигнала от любого трека (входящий пин с левой части ноды). В противном случае - только когда сигналы от всех входящих треков будут получены. Служит для настройки синхронизации логики, когда нужно какое-либо действие запустить когда все предыдущие завершились. Например, когда оба персонажа, управляемые мизансценой, дошли но конечных точек своих маршрутов

Когда гейт активируется, он, в соответствии с настройкой ActivationMode стартует дочерние треки. При этом, для всех входящих треков, которые соединены с ним через левый входящий пин и всё еще продолжают работу, вызывается прерывание работы текущей активной команды и завершение работы трека

Track

Трек является контейнером выполняемых команд. Каждый трек выполняет только одну активную команду до её завершения. После завершения последней команды в зависимости от настроек может послать сигнал выходному гейту или начать работать снова. Иконка трека идентифицирует режим работы соответственно настройкам



- Repeat - признак Continuous-трека, который никогда не завершает свою работу. После завершения работы всех команд начинает работу заново.Если в треке есть Continuous команда, то эта команда всегда продолжает работу и трек работа трека не стартует заново
- RandomOrder - проигрывать команды в произвольном порядке. Требует, чтобы все команды в треке были одного типа. В противном случае при валидации данных будет определена ошибка
- RandomizeOrderEachCycle - изменять порядок выполнения команд каждый цикл запуска трека
- RandomPlaySingleCommand - может выполняться только одна команда за цикл работы.
- Comment - текстовый комментарий в произвольной форме. Отображается на виджете ноде, может быть отредактирован прямо в виджете

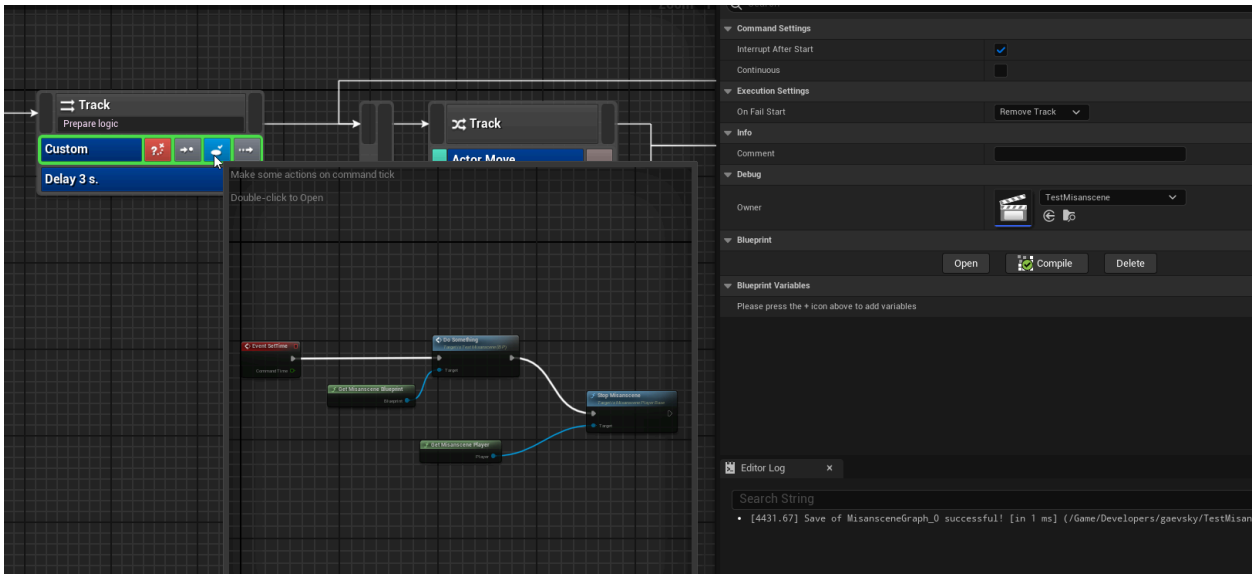
Настройки рандомизации могут использоваться, например, для проигрывания случайной анимации на персонаже из набора анимаций

Commands

Все команды имеют единственную общую для всех настройку поведения:

- OnFailStart - поведение на случай если при старте работы команды возникла ошибка и команда выполняться не может. Remove Track - родительский трек исключается из списка активных треков и прекращает работу, FinishTrack - трек принудительно завершает работу, не игравшиеся команды не выполняются, отсылает сигнал о о завершении внешним гейтам. SkipCommand - команда пропускается и начинает проигрываться следующая
- Comment - текстовый комментарий в произвольной форме. Отображается на виджете ноде, может быть отредактирован прямо в виджете
- Дочерний блюпринт для кастомизации поведения. Виджеты в панели Details позволяют Создавать/Открывать/Удалять блюпринт, создавать, копировать, вставлять новые переменные

Все команды соответствуют единому шаблону поведения. Самый общий случай реализован в команде UMisansceneCustomCommand



Каждая реализация класса пользовательской команды в проекте может расширять свое поведение специализированным классом блюпринта. Все типы команд могут использовать расширение своего функционала путем делегирования вызовов из c++ кода в дочерний блюпринт, который автоматически создается в редакторе по требованию. Нужные функции этого класса могут помечены специальными атрибутами, которые редактор интерпретирует как пользовательские расширяемые функции команды и отображает в виде кнопок с разными стилями, которые также задаются через мета атрибуты

Каждая команда должна определить свой жизненный цикл через реализацию функции IsContinuous - может ли команда завершится сама по своей внутренней логике или должна выполнятся, пока не завершат исполняющий её трек. Такая Continuous-команда может быть в треке только одна, и только последняя

Настройки CustomCommand

- Continuous - режим бесконечной работы команды. Не может быть прерван самой командой путем вызова из блюпринта функции Interrupt
- InterruptAfterStart - команда завершается сразу же после проверки возможности работы в функции CheckEntry и обработки события EventStart

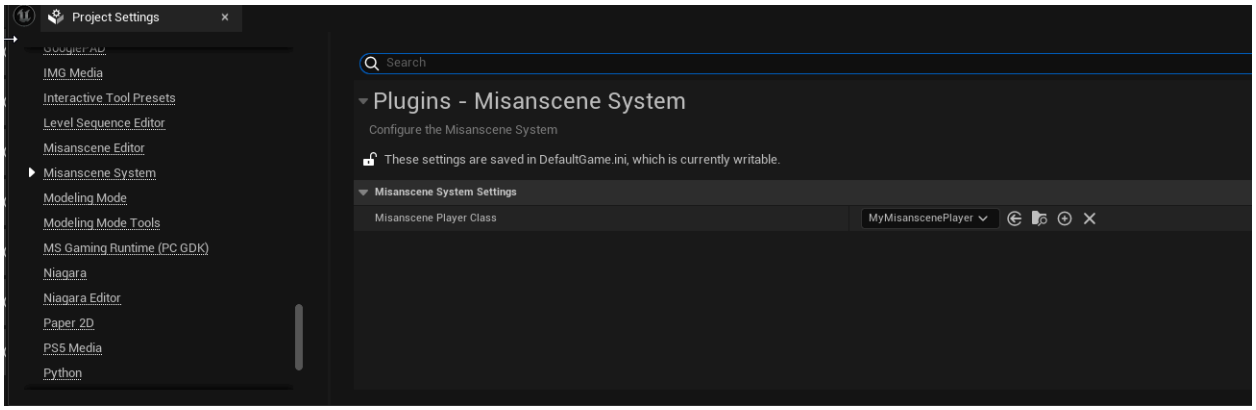
Каждый тик команда вызывает событие EventSetTime.
Если команда не является Continuous, то она может прервать своё действие в любой момент вызвав в функцию Interrupt. По окончании работы самой командой или внешней логикой, будет вызвано событие EventStop

Биндинг свойств команд

Мизансцена предоставляет встроенный механизм связывания переменных дочернего блюпринта мизансцены на специально отмеченные свойства классов пользовательских команд. Для этого требуется только сначала вручную создать блюпринт в корневой ноде мизансцены в панели Details. После этого все команды с такими размеченными свойствами для биндинга автоматически в панели Details будут отображать специальный виджет, который позволит автоматически создавать нужные типы переменных, выбирать переменные совместимых типов, удалять биндинг. Переименование и удаление связанных переменных так же поддерживается редактором.
Механизм биндинга рекомендуется использовать для создание переиспользуемых мизансцен. Заполнение значений связанных переменных происходит непосредственно на загруженной в редакторе карте

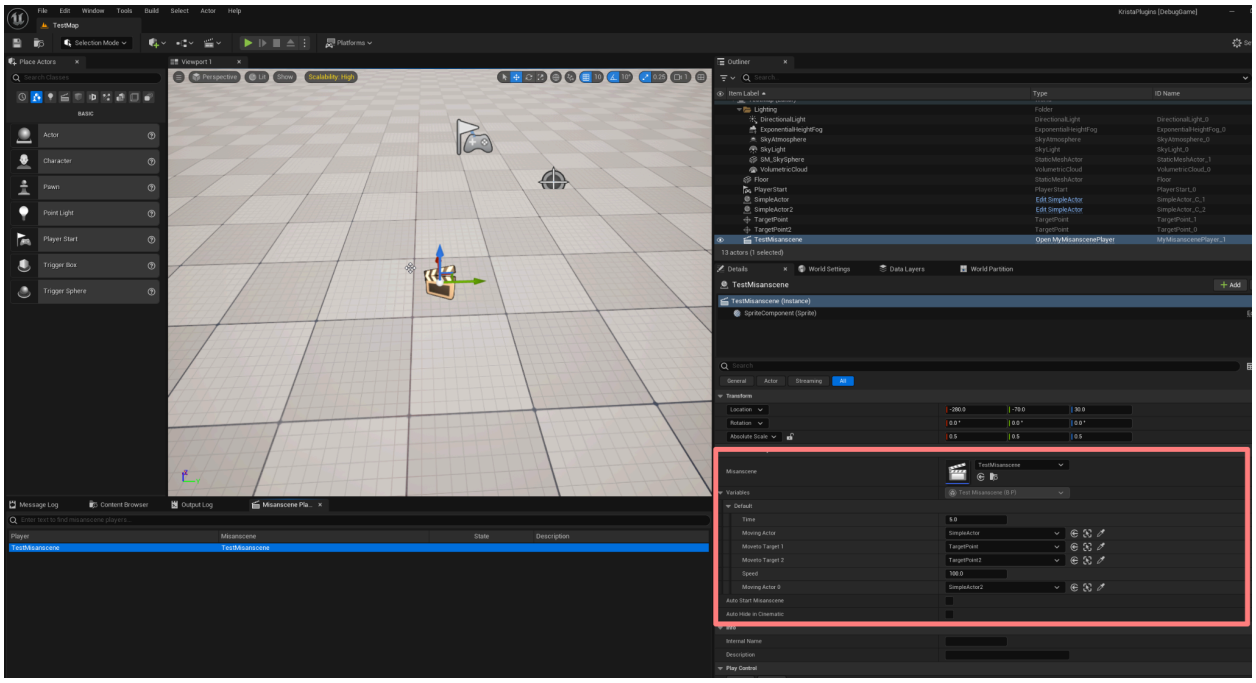
Проигрывание мизансцен

Мизансцена является шаблоном логики, которую проигрывает специальный актор на сцене - MisanscenePlayer. Собственную реализацию дочернего класса AMisanscenePlayerBase необходимо реализовать в проекте и указать в качестве класса плеера. Это требуется для автоматического создания плеера нужного типа



Базовый класс AMisanscenePlayerBase поддерживает сериализацию и восстановление своего состояния и состояния своих команд. Но сама логика вызова сохранения и загрузки лежит на стороне пользовательского кода
Количество MisanscenePlayer для одной мизансцены не ограничено. Каждый MisanscenePlayer при выполнении передаёт в собственную копию мизансцены настройки, которые являются Instanced переменными дочернего блюпринта мизансцены и могут быть использованы в любой дочерней команде через механизм биндинга свойств

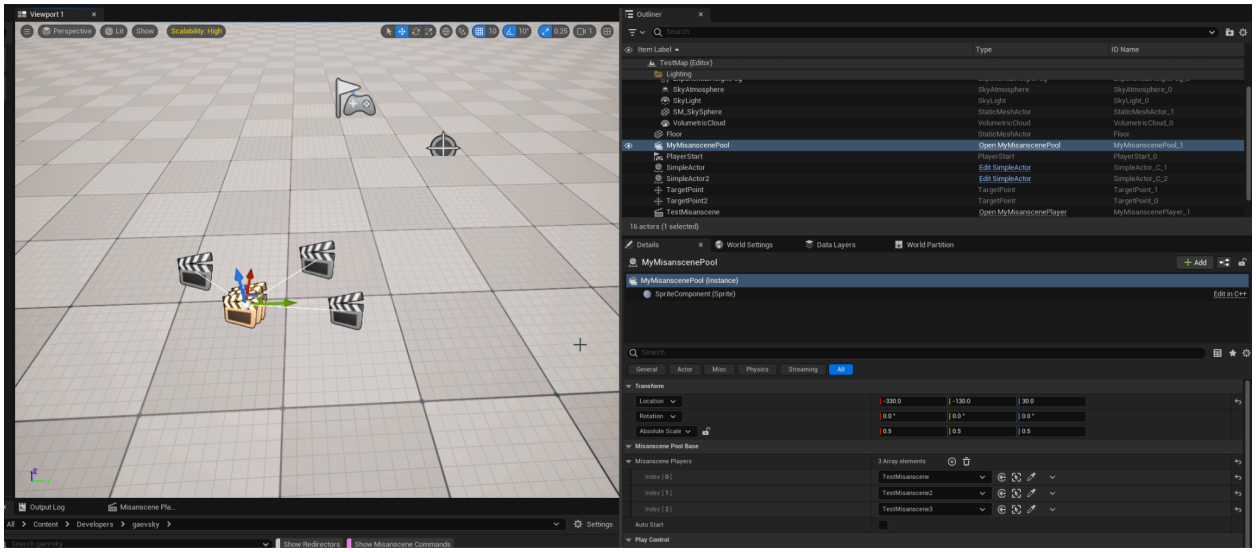
Чтобы создать MisanscenePlayer для конкретной мизансцены нужно ассет этой мизансцены перетащить Drag'n'Drop в окно ViewPort. Будет создан MisanscenePlayer с привязанной к нему мизансценой. Останется только заполнить значения переменных нужными данными



Шаблон мизансцены можно в любой момент поменять, нужные переменные будут обновлены автоматически

- AutoStartMisanscene - уастройка автозапуска. Мизансцена начнёт проигрываться сразу при вызове BeginPlay
- InternalName - Строка в произвольной форме, которая будет отображаться на сцене в редакторе над иконкой плеера для быстрого визуального поиска
- Description - описание в произвольной форме

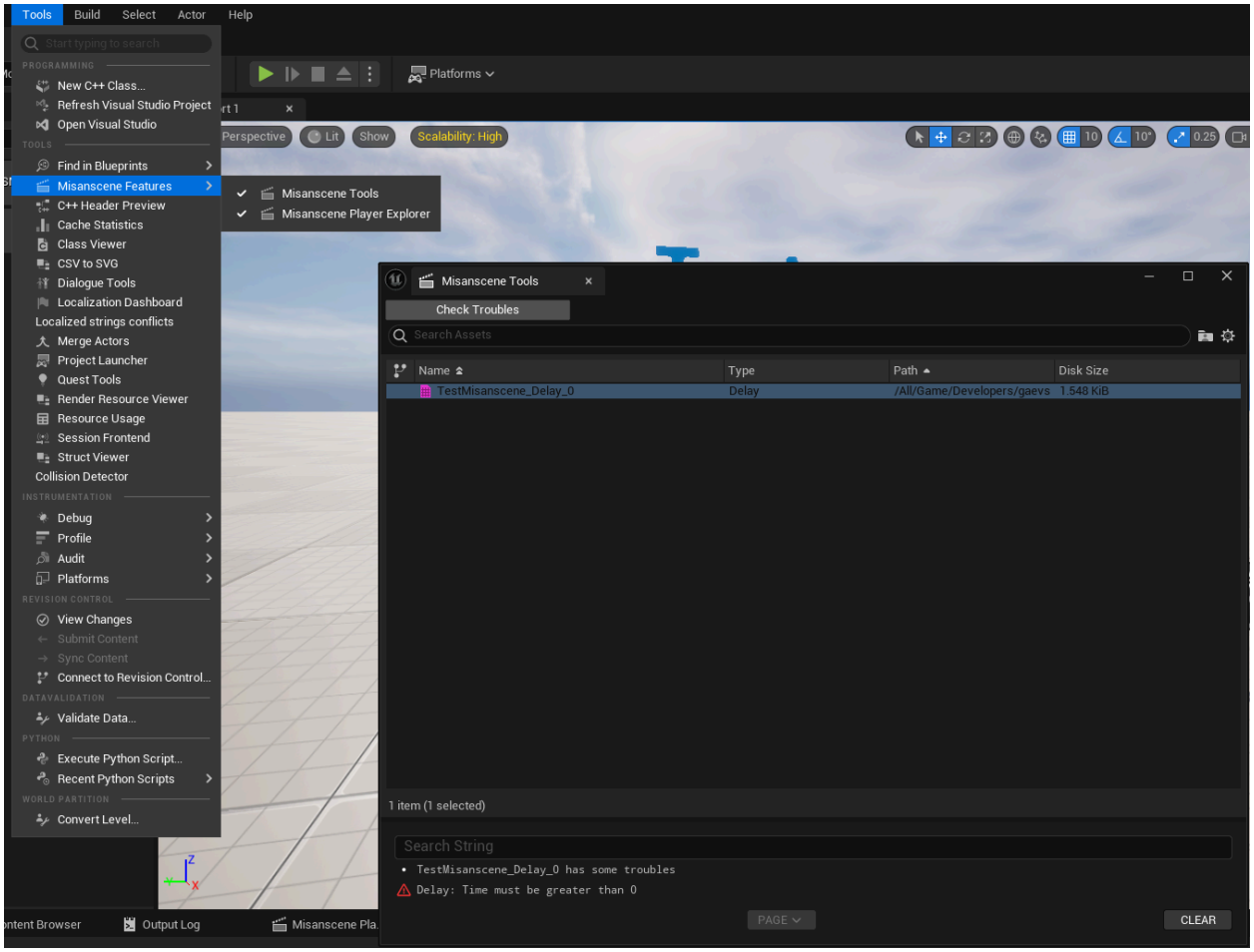
Также на стороне пользователя лежит задача создания класса MisanscenePool производного от AMisanscenePoolBase. Он позволяет связывать много MisanscenePlayer в один пул и Запускать/Останавливать их одновременно.



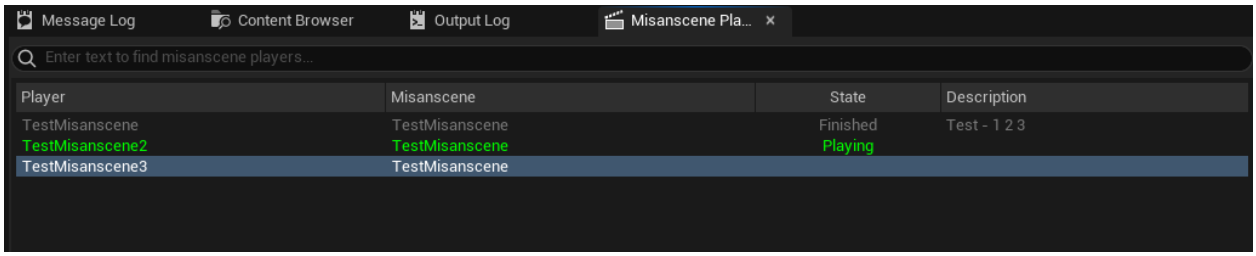
Дополнительные функции

В плагине поставляется несколько вспомогательных классов, которые предоставляют дополнительный сервис в редакторе:

- MisansceneTools - форма для массовой проверки ассетов мизансцен на целостность с визуализацией ошибок и возможностью открыть ошибочный ассет для исправления или удалить его.



- MisansceneExplorer - стыкуемый в произвольное место виджет редактора Unreal, который отображает список всех акторов MisansценePlayer, которые есть на текущем загруженном уровне. Позволяет оперативно находить нужные акторы MisansценePlayer в окне Outliner и открывать редактор для конкретной мизансцены. А также отображает статус работы конкретного MisansценePlayer в режиме PlayInEditor



- FFrontendFilter_MisansceneCommandVisibility - фильтр для Content Browser, позволяющий управлять видимостью ассетов команд. По умолчанию ассеты команд скрыты
- FGameplayDebuggerCategory_Misanscene - вкладка для GameplayDebugger, которая в игре показывает состояние всех доступных MisansценePlayer на текущем

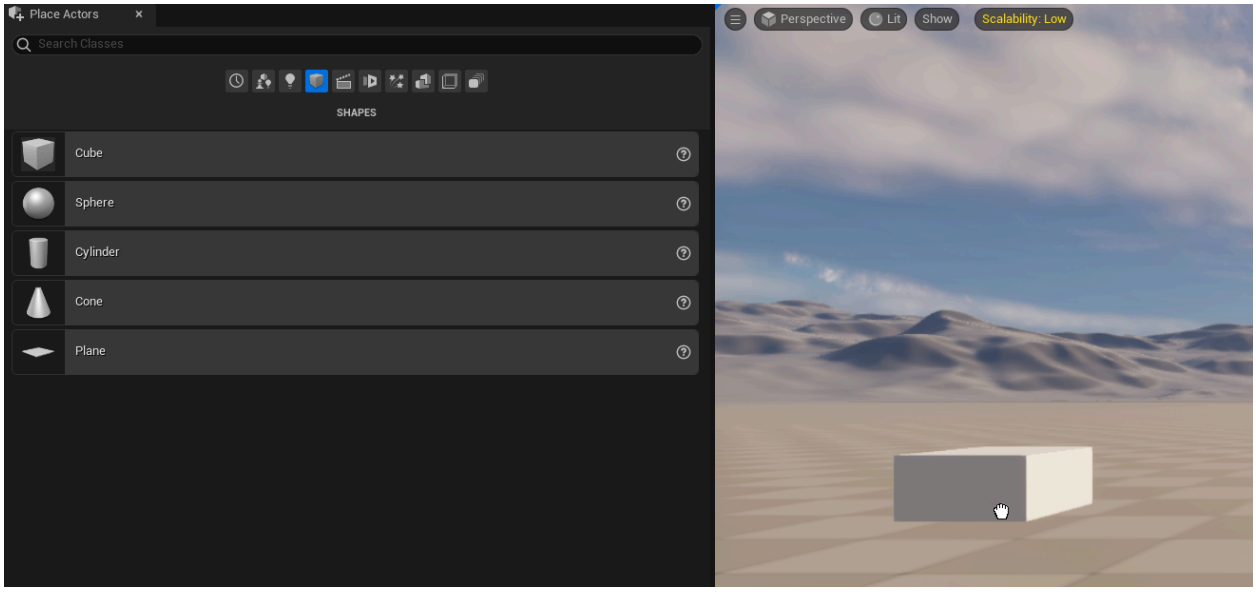
уровне, а также помечает всех акторов, которые используются в работающих MisanscenePlayer-ах (для этого требуется реализация собственных команд)

Проверка работоспособности плагина

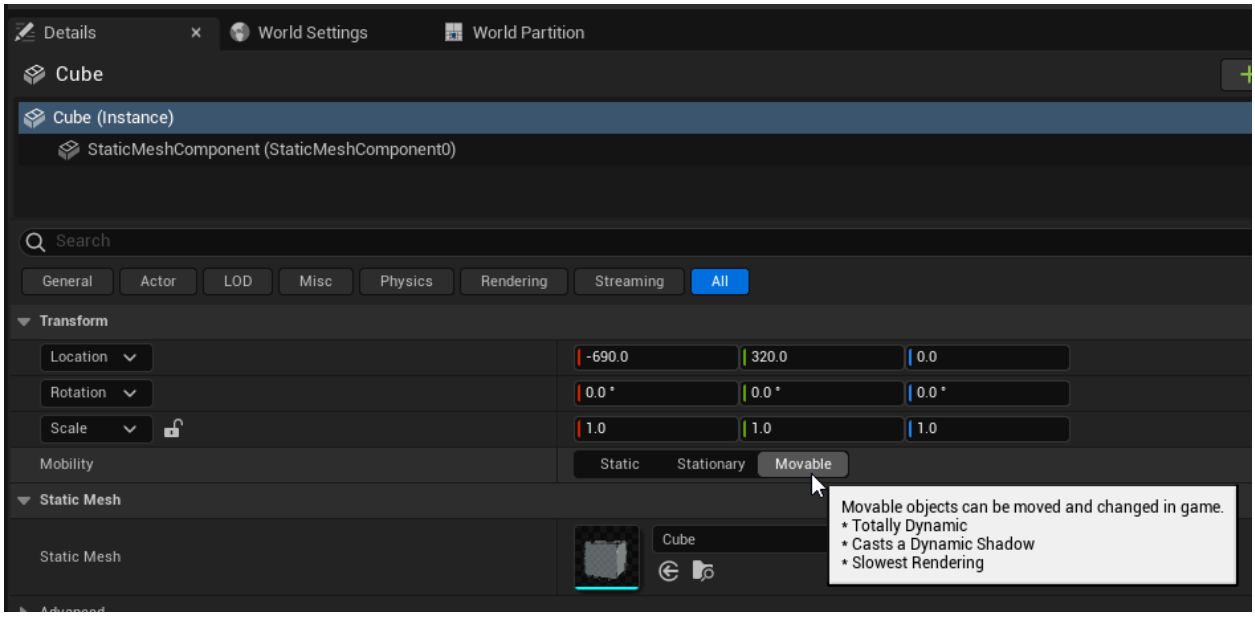
Действия совершаются в новом пустом проекте Unreal (Empty Project - C++) после установки плагина с зависимостями.

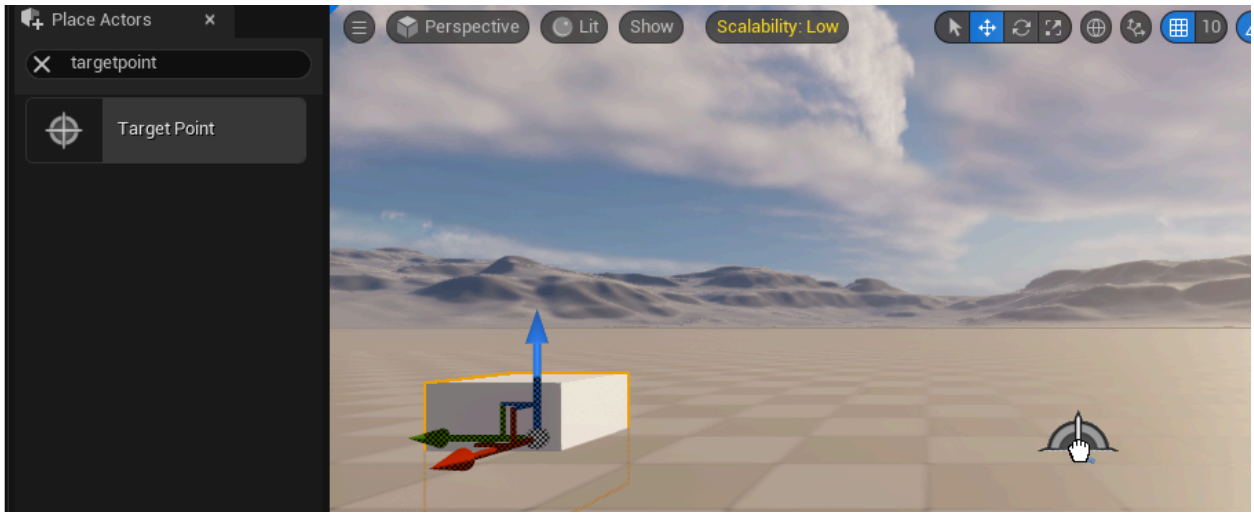
Создание мизансцен

- В окне Place Actors (выбрать в меню Window/Place Actors если оно скрыто) выбрать складку Shapes и перетащить актора Cube в игровой уровень (так, чтобы его было видно на камере).

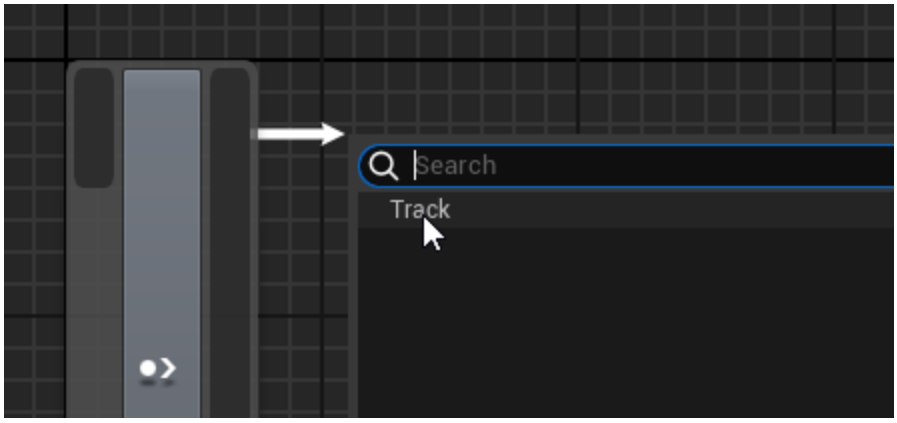


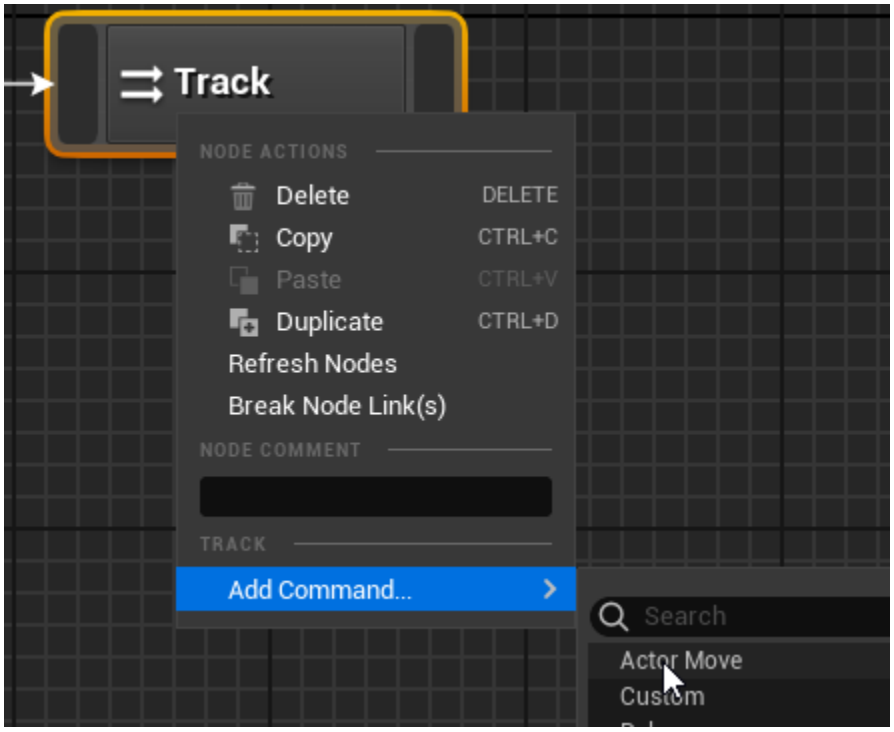
- Выбрать этот актор в уровне. В окне Details найти пункт Mobility и переключить на Movable

- 

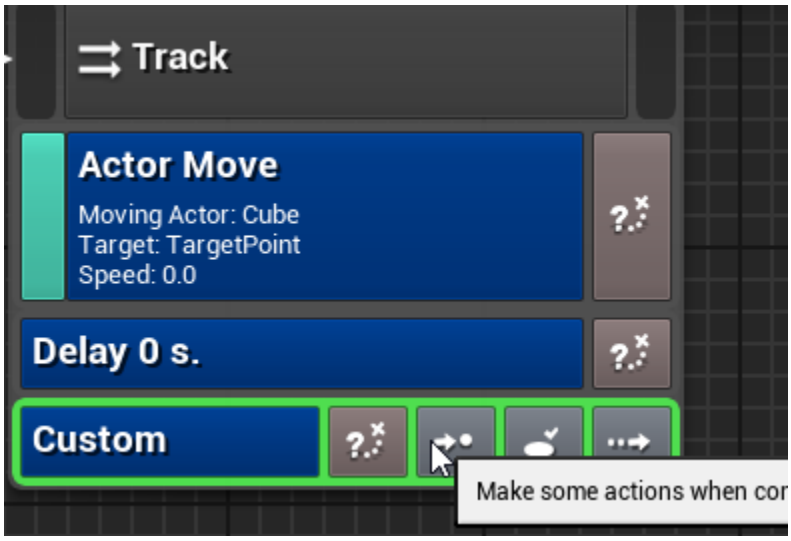
- Ввести в строке поиска в окне Place Actors строку “TargetPoint” и перетащить актора TargetPoint в игровой уровень, в стороне от куба из предыдущего пункта
- 

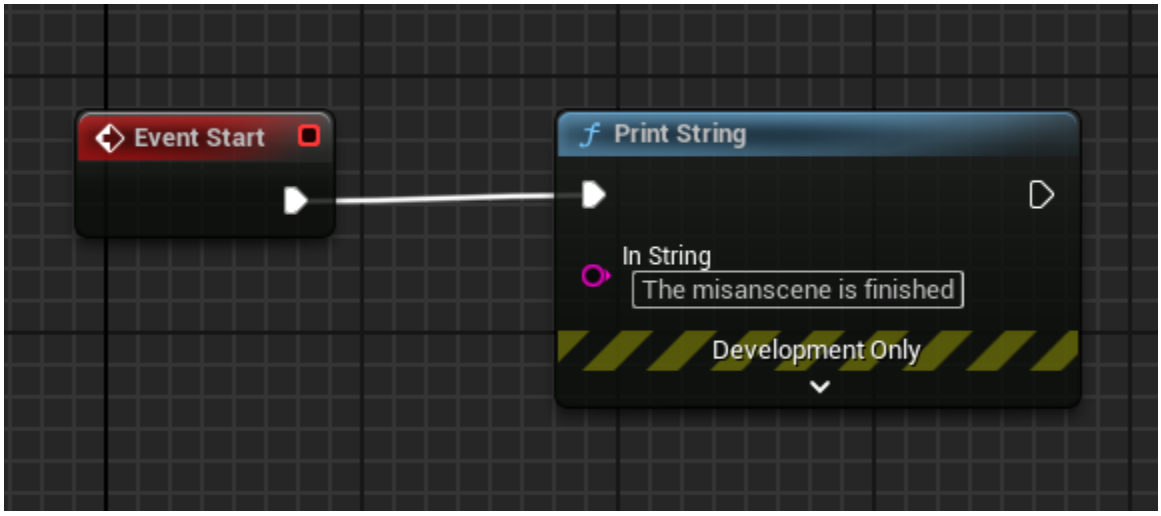
- Создать новую папку внутри Content с именем TestMisanscene
- Открыть её в Content Browser, в контекстном меню по правому клику выбрать Misanscene System/Misanscene
- Ввести имя нового ассета (TestMisansceneRoot)
- Двойным кликом по ассету открыть редактор мизансцен
- В открывшемся редакторе будет виден корневой узел мизансцены. Зажать левую кнопку на полоске в правой части этого узла и перетащить вправо. В открывшемся меню выбрать Track

- 

- Кликнуть правой кнопкой мыши по созданному узлу трека, выбрать в меню Add Command -> Actor Move
- 

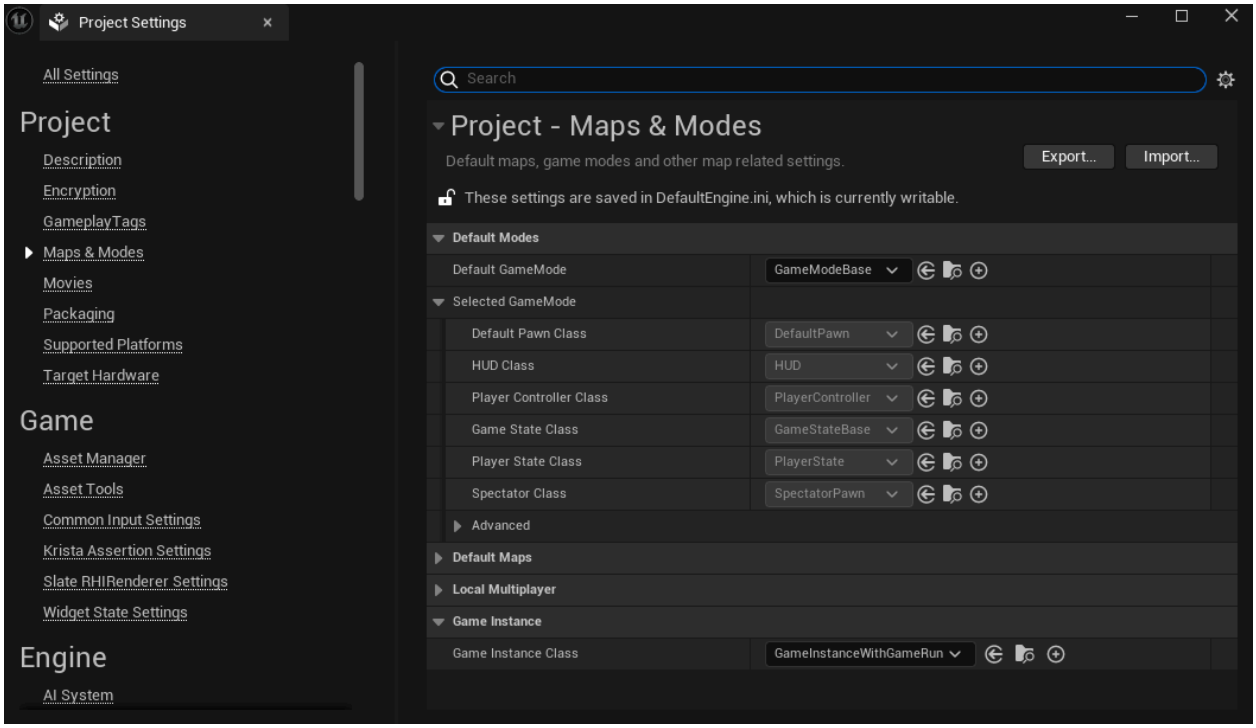
- Кликом выбрать созданную команду. В окне Details в правой части редактора указать:
- В строке Moving Actor - поставленный в начале куб (по умолчанию называется Cube)
- В строке Move To Target - один из поставленных акторов Target Point (по умолчанию называется TargetPoint)
- В строке Speed - 100
- Аналогично предыдущему пункту, добавить через контекстное меню команду Delay
- Выбрать команду, в панели Details ввести 1 в строке Time
- Аналогично предыдущему пункту, добавить через контекстное меню команду Custom

- Двойным кликом по кнопке Start (с изображением стрелки и точки) открыть редактор блюпринта команды
- 

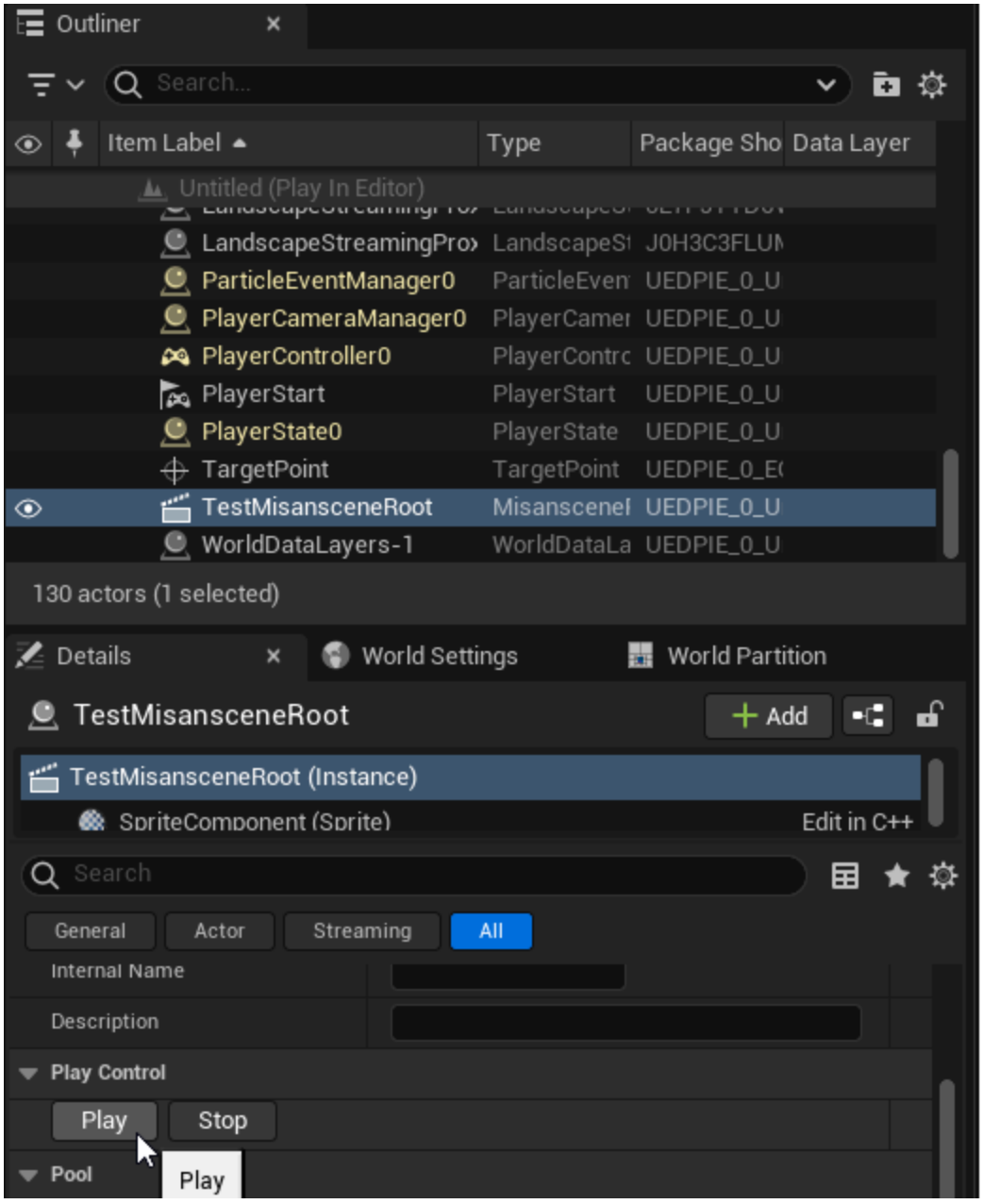
- Добавить следующий блюпринт
- 

Запуск мизансцены в игре

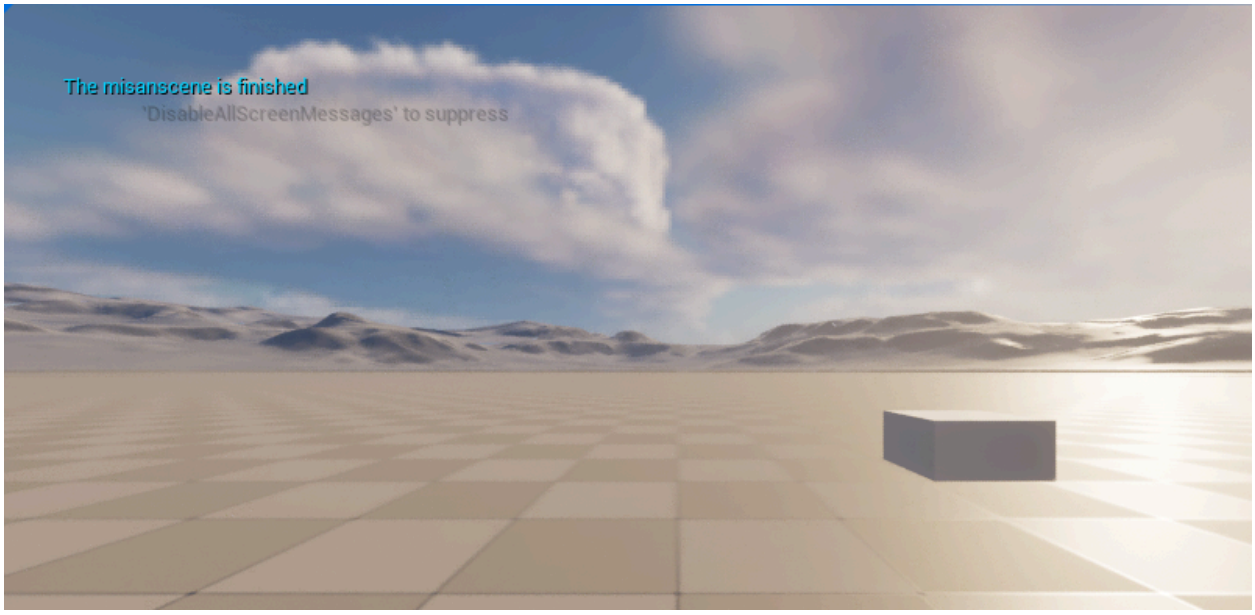
- Открыть меню Edit/Project Settings/Maps & Modes и выбрать в качестве GameInstance класс GameInstanceWithGameRun



- Ассет TestMisansceneRoot перетащить из Content Browser в окно игры. При этом будет автоматически создан актор с именем TestMisansceneRoot в уровне
- Начать игру
- Найти в Outliner актор TestMisansceneRoot из предыдущего пункта и выбрать его
- В окне Details найти кнопку Play и нажать её



- Удостовериться, что куб начал движение к финальной точке
- После завершения движения, через одну секунду, на экране и в логе появится надпись The misanscene is finished

- 
The misanscene is finished
'DisableAllScreenMessages' to suppress

EtudeSystem

Плагин EtudeSystem предоставляет ресурсы, редактор и базовую реализацию системы игровой логики на основе этюдов - иерархически связанных объектов, описывающих игровые события и состояния мира игры, в которых может быть реализована произвольная логика, специфичная для конкретной игры.

Плагин требует наличия плагинов KristaMisc и KristaAssertions для работы.

Как установить плагин в проект

1. Установить плагины KristaMisc, KristaAssertions и GameRun по их инструкции.
2. Перенести директорию EtudeSystem в директорию Plugins проекта.
3. Запустить редактор
4. Перейти в меню Edit -> Plugins. Найти плагин EtudeSystem и поставить галку напротив
5. Перезапустить редактор

Состав плагина

В плагине поставляются классы компонентов этюдов которые реализуют необходимый базовый функционал для построения произвольной исполняющей логики - ULogicComponent и UEtudeBracketComponent. На основе этих классов пользователь может реализовать свои собственные классы со специфичной для конкретного случая логикой. Например, триггеры на какие-либо игровые события.Все классы компонент, реализованные к конкретном проекте, автоматически становятся доступны в редакторе этюдов.

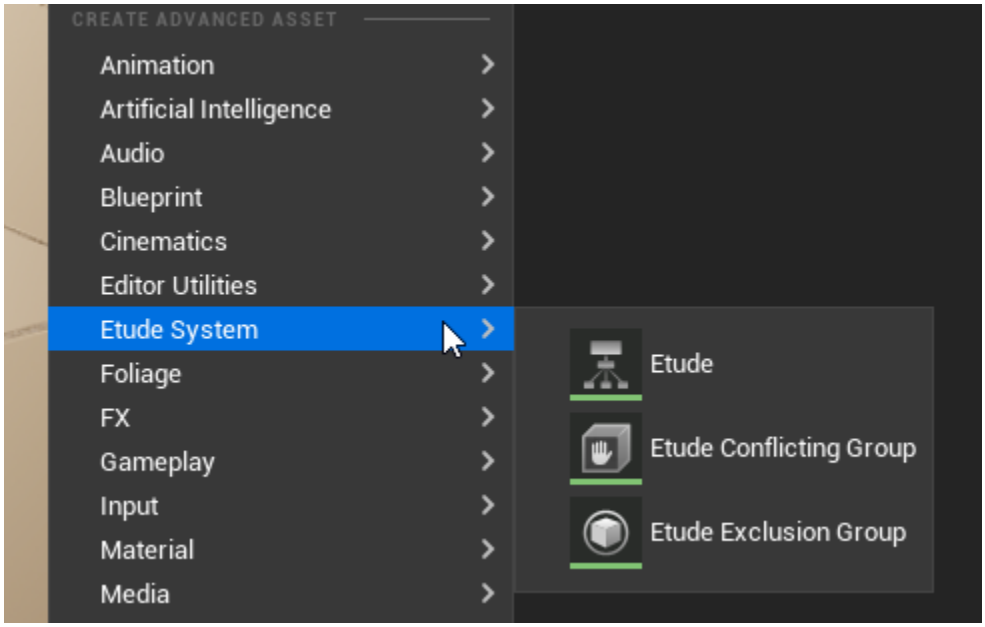
Система этюдов является глобальной GameRun-системой, работает независимо от запущенного уровня. Состояние любого этюда можно определить или изменить в любой момент из c++ кода или из любых блюпринтов

Система этюдов состоит из следующих типов ассетов:

- Etude - Этюд, контейнер игровой логики, содержащий в себе ссылки на дочерние или одноранговые этюды, настройки своей активации, а также на компоненты
- Component - компонент этюда, который реализует конкретную игровую логику в период активности этюда
- EtudeConflictingGroup - конфликтная группа. Среди этюдов входящих в одну конфликтную группу может быть активен только этюд с самым высоким приоритетом

- EtudeExclusionGroup - группа исключения. Если один из этюдов, входящих в группу исключения, деактивируется, то все этюды из этой группы также деактивируются

Все типы ассетов кроме компонент этюдов могут создаваться в ContentBrowser через контекстное меню. Ассеты компонент создаются исключительно в редакторе этюдов. Удаление ассетов этюдов и компонентов допустимо только из редактора этюдов, который позволяет корректно разорвать ссылки между удаляемыми и оставшимися ассетами. По умолчанию ассеты команд скрыты в ContentBrowser. Для их отображения есть настройка Edit -> Project Settings -> EtudeEditor -> Show Command Assets и фильтр для ContetnBrowser, который может управлять их видимостью для пользователя.

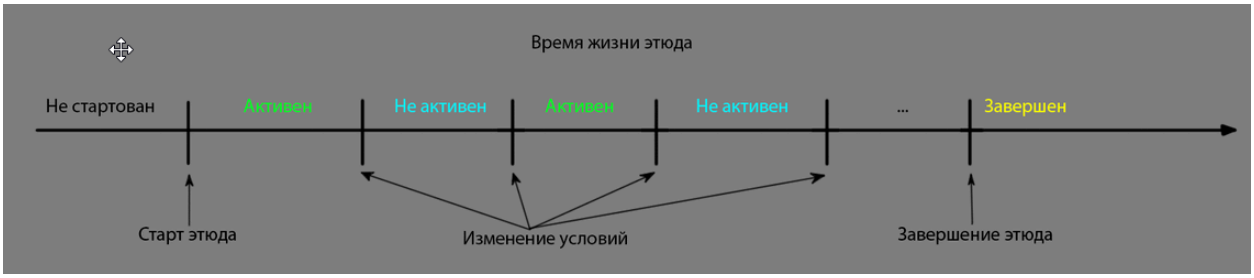


Жизненный цикл этюдов

Этюд в процессе игры имеет свой четко определенный жизненный цикл и может находиться в одном из 5 состояний:

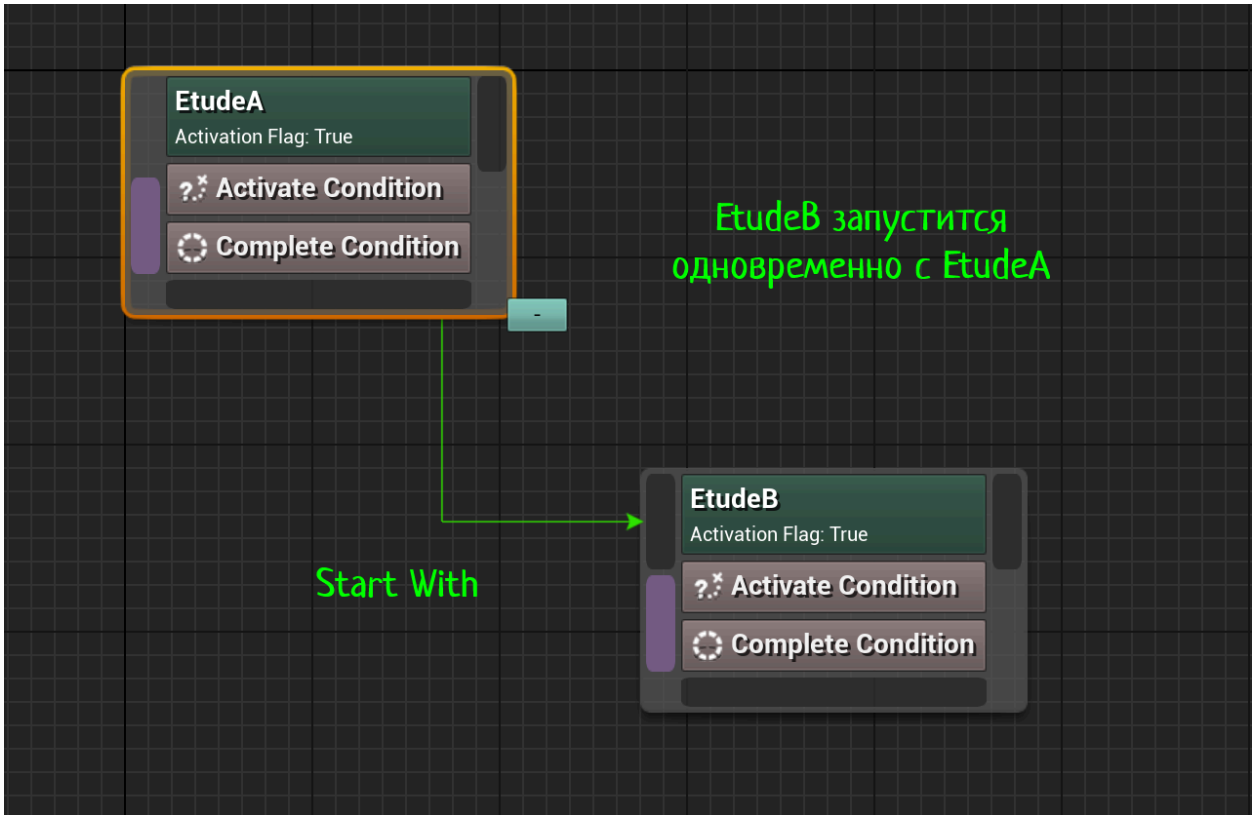
- Не стартован
- Стартован и активен
- Стартован и неактивен
- Ожидание завершения (себя и дочерних этюдов)
- Завершен

Этюд, который перешел из состояния “не стартован” в любое другое - уже никогда не может перейти в состояние “не стартован”. Этюд, перешедший в состояние “завершен” - уже никогда не может перейти в любое другое состояние. Стартованный этюд может сколько угодно раз переходить в состояние “Активен” и “Не активен” и обратно



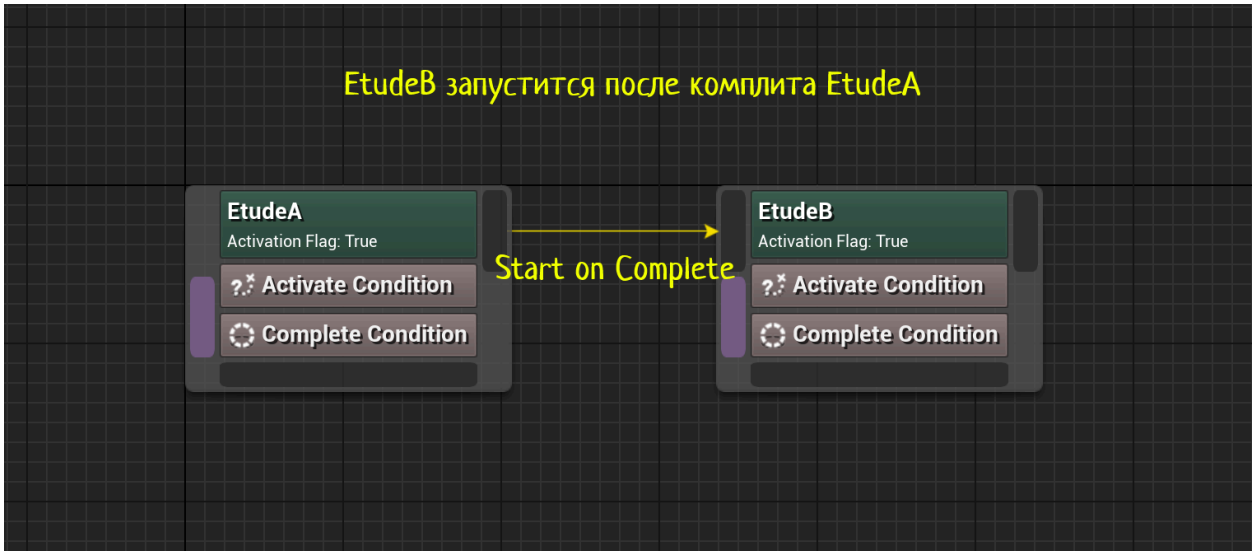
Между этюдами существует строгая иерархическая связь типа “родитель - ребенок”. Этюд-ребенок может быть активен только тогда, когда активен его этюд-родитель. Два этюда, имеющие одного родителя называются сиблингами.
Также между этюдами существует функциональная связь двух видов:

- Start With. Этюды, указанные в Start With этюда A стартуют одновременно со стартом этюда A. В Start With этюда A можно указывать только те этюды, которые являются дочерними к этюду A, либо имеют с ним общего родителя.



- Start on Complete. Этюды, указанные в Start on Complete этюда A стартуют при завершении этюда A. В Start on Complete этюда A нельзя указывать этюды,

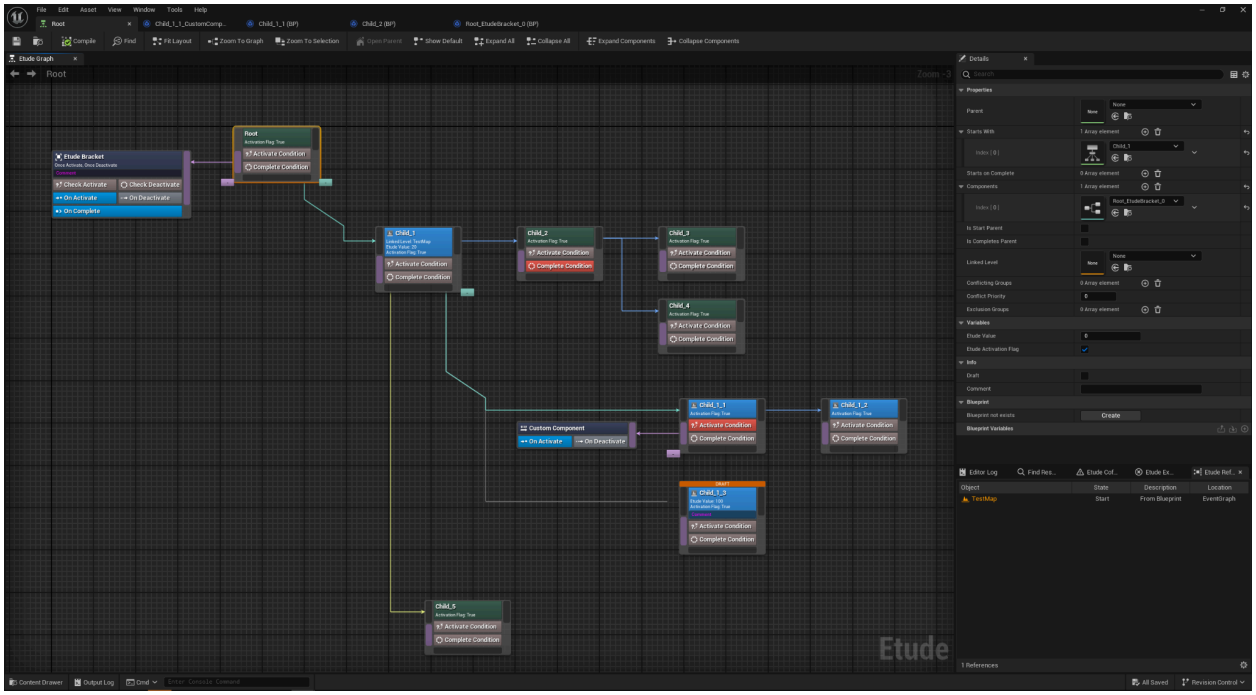
дочерние этюду A



Редактирование этюдов

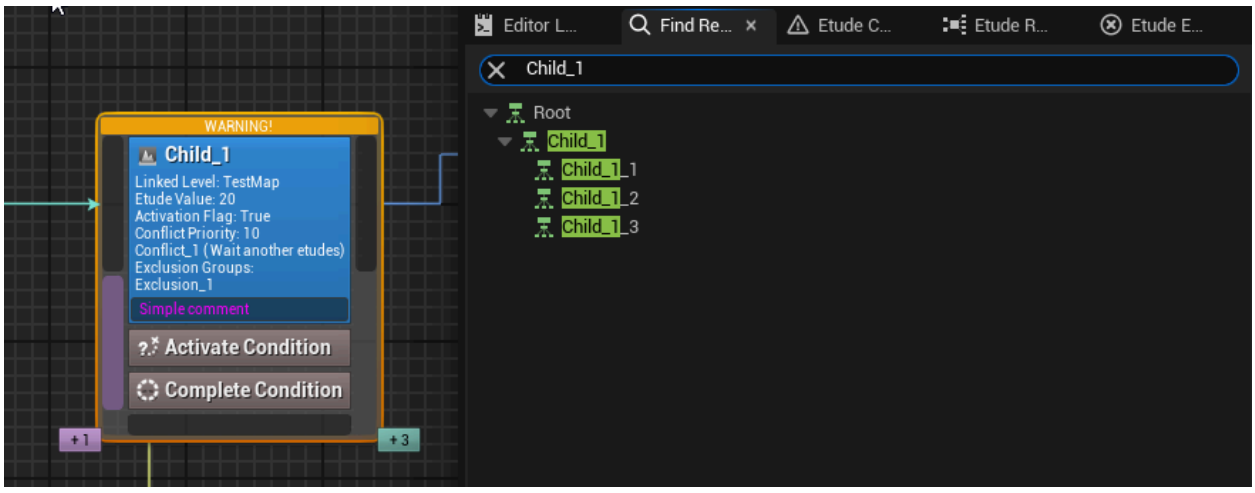
Двойной клик на любой ассет этюда или его компоненты открывает специализированный редактор, в котором автоматически строится дерево этюдов начиная с выбранного и на определённую глубину, которая может меняться в настройках проекта.Редактор позволяет создавать новые этюды и компоненты, менять связи между ними с помощью drag&drop'a, удалять этюды и компоненты.
При сохранении открытых в редакторе ассетов автоматически проводится валидация настроек и компиляция блюпринтов. Результаты выводятся в лог, все проблемные ноды

подсвечиваются

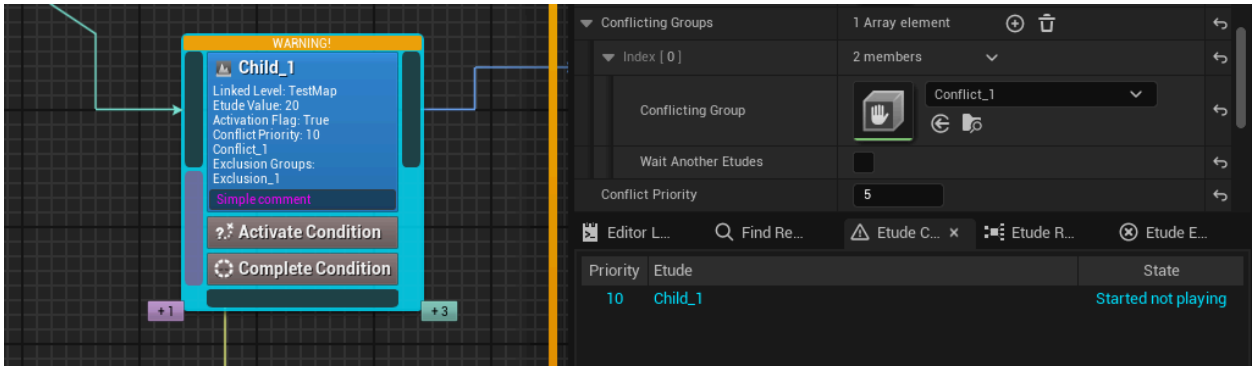


Редактор имеет следующие панели:

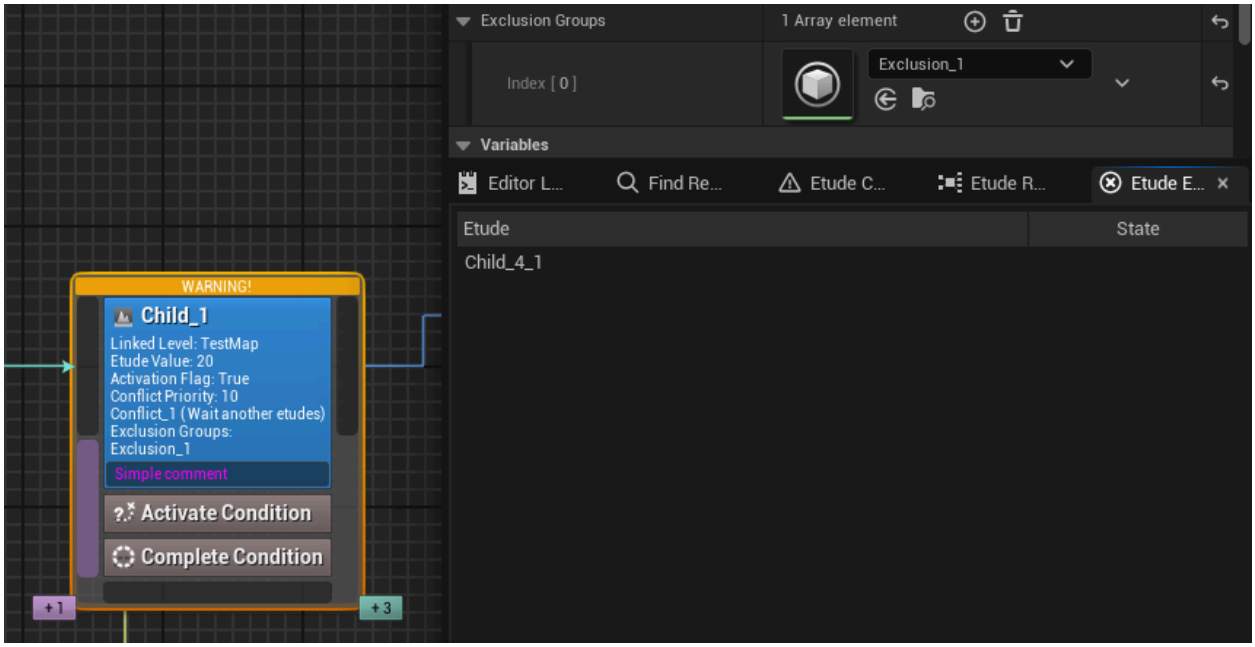
- Панель инструментов. Предназначена для операций поиска и сохранения ассетов, компиляции дочерних блюпринтов, массовые операции над открытым деревом этюдов - выравнивание по сетке, сокрытие/раскрытие дочерних элементов и проч.
- Details - панель для редактирования свойств выделенного на графе этюда или компоненты
- Log - окно лога сохранения и компиляции, которое отображает все возникшие ошибки с подсветкой соответствующих нод на графе
- Find Results - окно поиска по имени этюда. Поиск происходит по всем существующим этюдам. Из окна можно открыть интересующий этюд



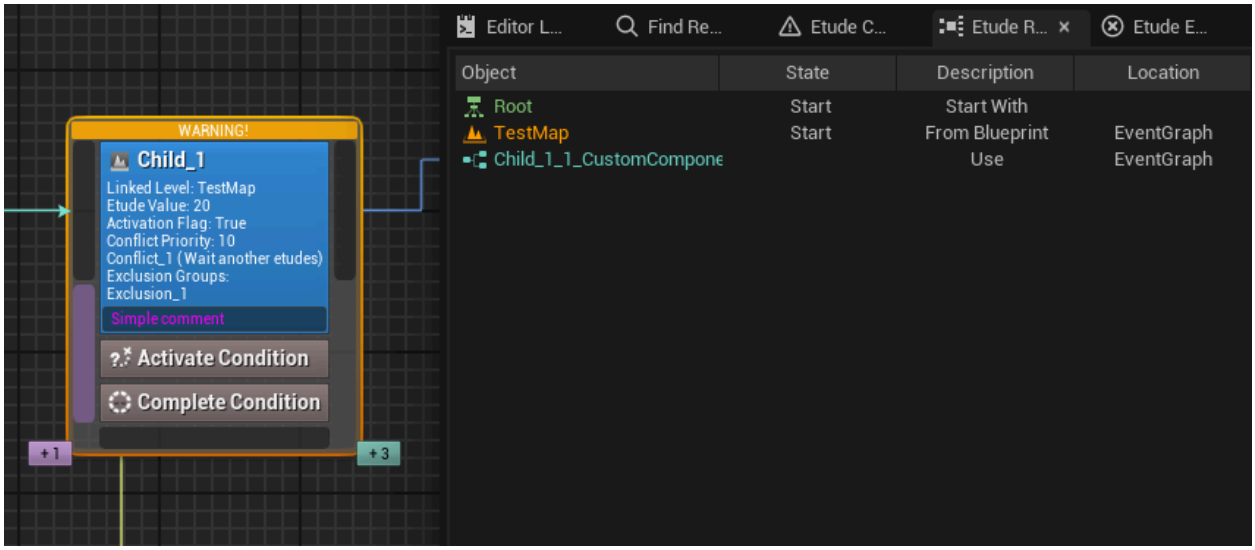
- Etude Conflicts - отображает для выбранного этюда все этюды входящие с ним в одну конфликтную групп. В режиме Play in Editor отображает их состояние



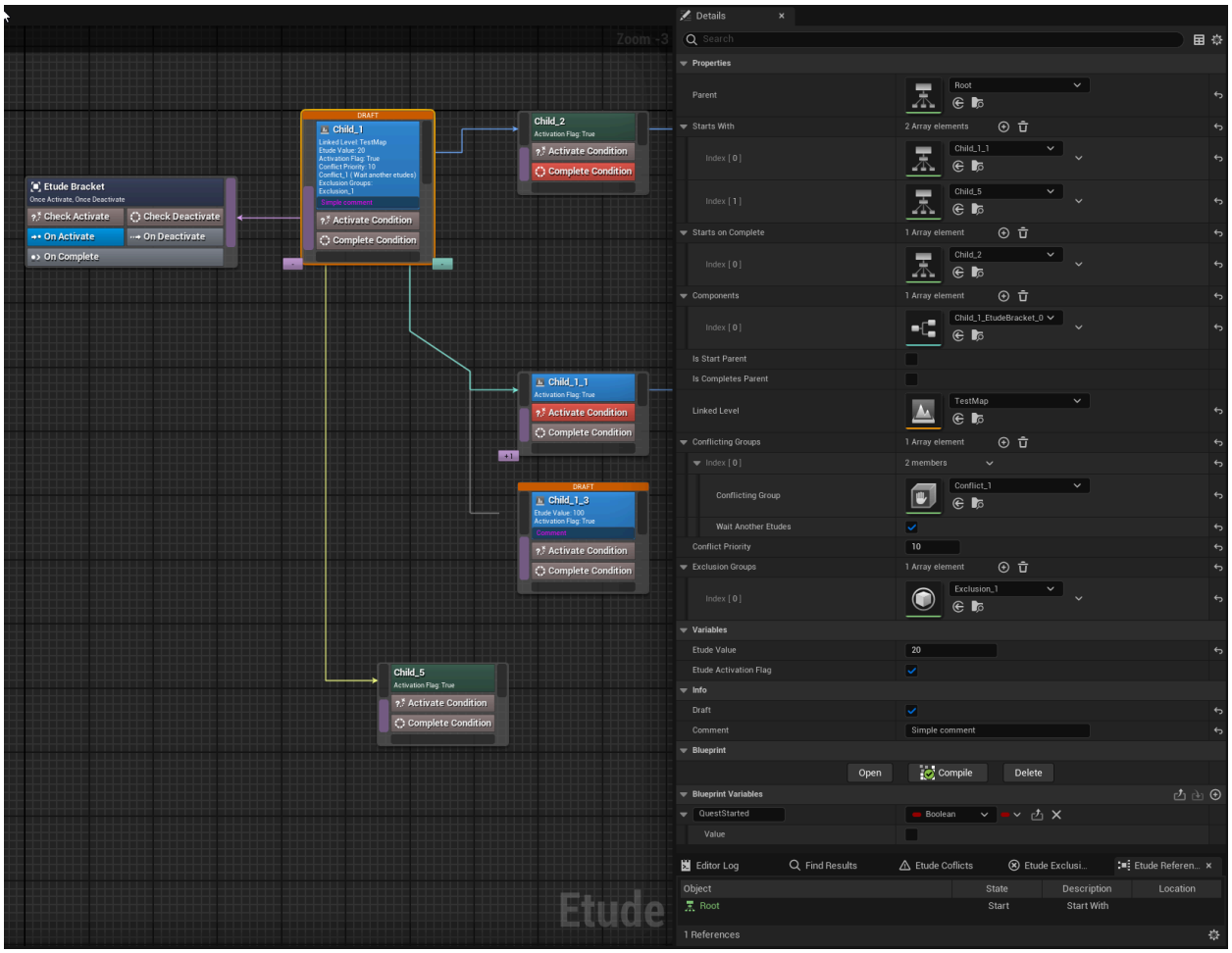
- Etude Exclusions - отображает для выбранного этюда все этюды входящие с ним в одну группу исключения. В режиме Play in Editor отображает их состояние



- Etude References - отображает все сущности в проекте, которые каким-либо образом ссылаются на выбранный этюд и каким образом его используют


Editor L...
Find Re...
Etude C...
Etude R...
Etude E...
Object
State
Description
Location
Root
Start
Start With
TestMap
Start
From Blueprint
EventGraph
Child_1_1_CustomCompone
Use
EventGraph
WARNING!
Child_1
Linked Level: TestMap
Etude Value: 20
Activation Flag: True
Conflict Priority: 10
Conflict_1 (Wait another etudes)
Exclusion Groups:
Exclusion_1
Simple comment
Activate Condition
Complete Condition
+1
+3

Свойства этюда

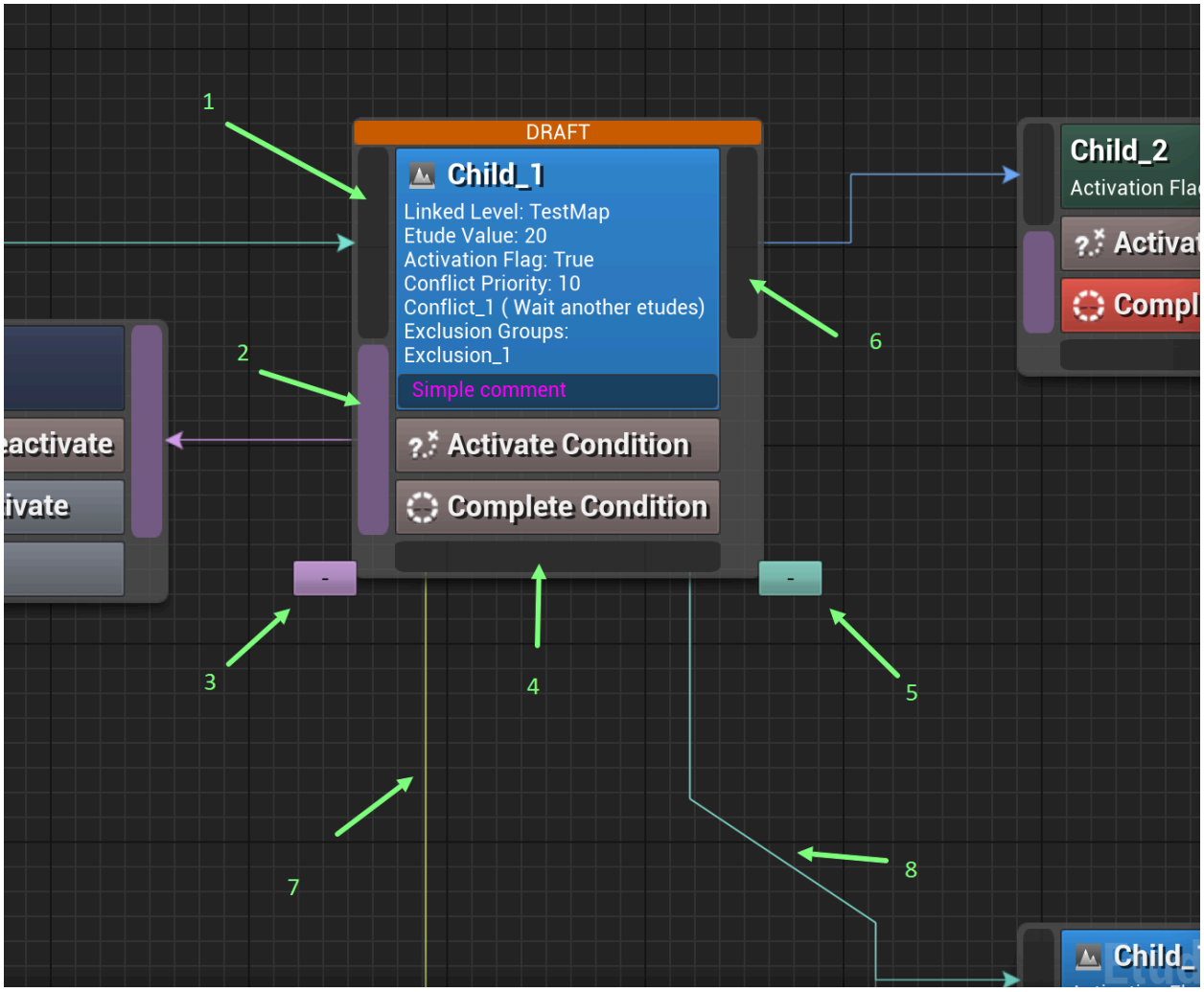


- Parent - ссылка на родительский этюд
- Start With - ссылки на этюды, стартующие вместе с текущим этюдом
- Start on Complete - ссылки на этюды, стартующие при завершении текущего этюда
- Components - ссылки на компоненты этюда
- Is Start Parent - при старте этого этюда должен стартовать родительский этюд
- Is Complete Parent - при завершении этого этюда должен завершаться родительский этюд
- Linked Level - ссылка на игровой уровень. Запущенный этюд при загрузке указанного уровня автоматически активируется и деактивируется при загрузке другого уровня. Все дочерние этюды автоматически ведут себя также. Заголовок ноды отдельно подсвечивается
- Conflicting Group - ссылки на конфликтные группу
- Wait Another Etude - этюд не запустится пока активен хотя бы один этюд с такой же конфликтной группой

- Conflict Priority - приоритет запуска этюда при наличии конфликтующих активных этюдов
- Exclusion Groups - группы исключения этюда
- Специальные переменные
  Etude Value - целочисленный параметр, который можно использовать произвольным образом. Доступ возможен как из кода так и через блюпринты
  Etude Activation Flag - параметр, который является неявным условием активации этюда (default = true). Этот параметр всегда используется вместе с условиями активации, которые могут быть реализованы в блюпринте этюда - кнопка Activate Condition на ноде этюда
- Draft - флаг помечает этюд как содержащий временную логику и несет только информационную нагрузку. Такие этюды отмечаются специальной плашкой наверху. Если в процессе сохранения и компиляции такого этюда возникают Warning-и, то они не отображаются ни в логе, ни над нодой в графе
- Comment - комментарий в произвольной форме
- Набор виждетов управления дочерним блюпринтом. Позволяют Создавать/Открывать/Удалять блюпринт, создавать, копировать, вставлять новые переменные

Нода этюда

Нода этюда на графе имеет следующие активные элементы



1. Входной пин для отображения связи с родительским этюдом в его выходной пин 4 (Start With) или с сиблингом в его выходной пин 6 (Start On Complete)
2. Выходной пид для отображения связей с компонентами этюда
3. Кнопка сворачивания/разворачивания дочерних компонент
4. Выходной пин Start With для подключения дочерних этюдов или сиблингов
5. Кнопка сворачивания всех дочерних этюдов
6. Линк до сиблингов запускаемых Start On Complete
7. Линк до сиблингов запускаемых Start With
8. Линк до всех дочерних этюдов как автоматически запускаемых в StartWith, так и запускаемых сторонней логикой (незавершённые линки)

На самой ноде этюда есть 2 кнопки для специализированных функций дочернего блюпринта. Двойной клик открывает эту функцию в блюпринте

- Activate Condition - вызывается каждый раз при обновлении всего дерева этюдов. Ее результат вместе со значением EtudeActivationFlag активирует или деактивирует этюд. Если функция не реализована, то по умолчанию возвращается true

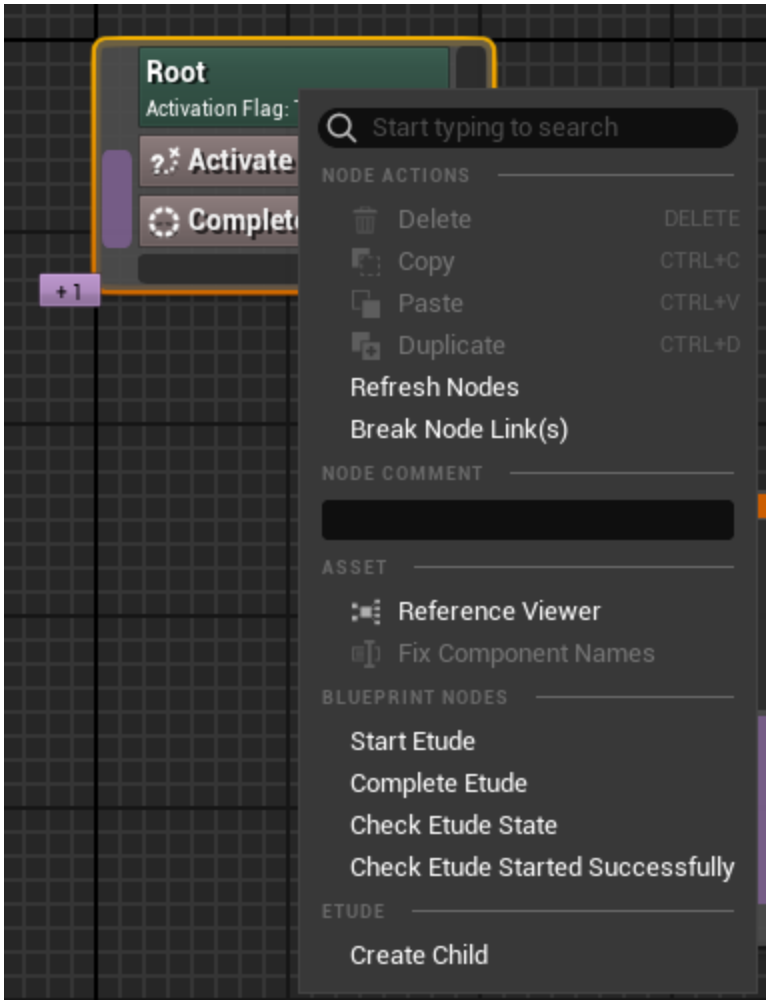
- Complete Condition - условие при котором этюд может быть завершен. Если возвращает false, то при попытке завершить этюд он перейдет в состояние “ожидание завершения”.Если функция не реализована, то по умолчанию возвращается true

Создать новый этюд можно как из контекстного меню на графе, так и вытянув линк из пина 4 (StartWith) или 6(StartOnComplete). Создание нового этюда будет происходить через диалог выбора папки ассета. Если этюд создается из пина 4, то будет создан дочерний этюд, если из пина 6, то будет создан сиблинг. Диалог предложит создать новый его в папке родительского этюда и ввести новое имя согласно шаблону

Новые компоненты создаются вытягиванием линка из пина 2(Components)

После создания рекомендуется запустить Fit Layout на панели инструментов или нажать Ctrl + Space

Контекстное меню ноды этюда содержит пункты, которые позволяют в буфер обмена поместить код создания блюпринтовых функций, в который указан выбранный этюд, и вставить эту функции в любом блюпринте

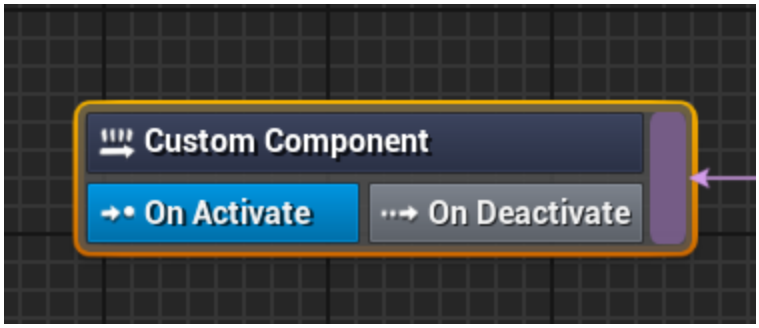


Ноды стандартных компонент

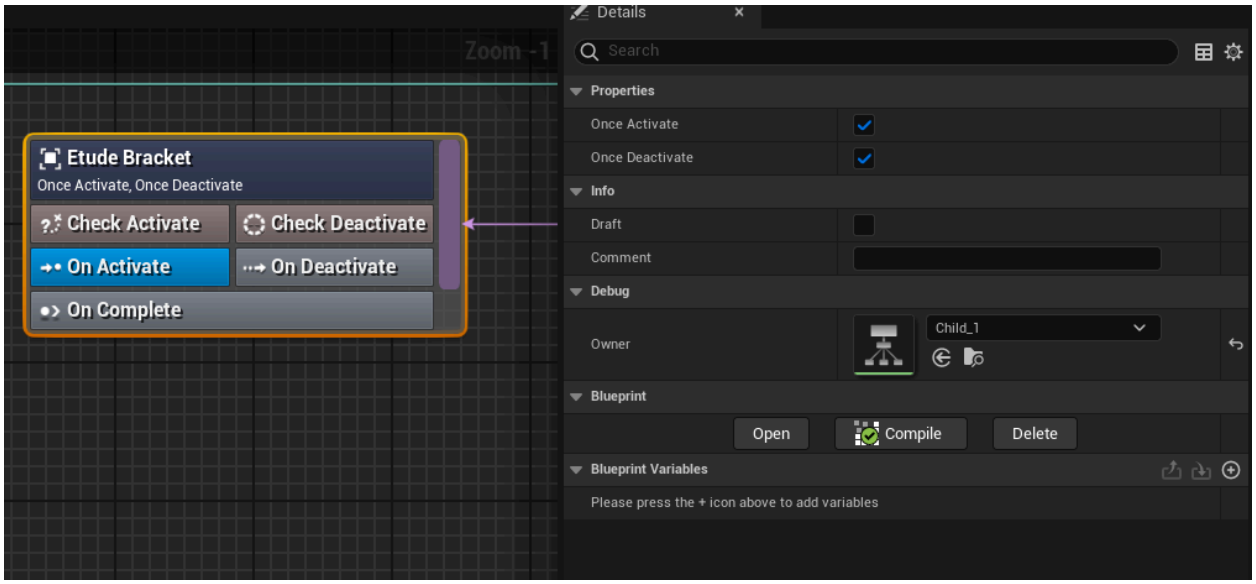
У всех классов компонент есть общие для всех свойства

- Draft - флаг помечает компоненту как содержащую временную логику и несет только информационную нагрузку. Такие компоненты отмечаются специальной плашкой наверху. Если в процессе сохранения и компиляции такой компоненты возникают Warning-и, то они не отображаются ни в логе, ни над нодой в графе
- Comment - комментарий в произвольной форме

Все компоненты происходят от базовой го класса ULogicComponent, которая в редакторе представлена как Custom Component. Она позволяет реализовать пользовательскую логику как реакцию на активацию и деактивацию родительского этюда в соответствующих методах дочернего блюпринта OnActivate и OnDeactivate



Компонента EtudeBracket является расширением компоненты ULogicComponent и попозволяет подключить в дочернем блюпринте пользовательскую логику с дополнительными проверками на все события жизненного цикла этюда - активацию, деактивацию. А так же реакцию на завершение этюда



- Once Activate - если компонента ни разу не отрабатывала логику активации, была активирована и функция CheckActivate возвращает true, то компонента сохранит признак активации и больше никогда не будет обрабатывать эту логику
- Once Deactivate - если компонента ни разу не отрабатывала логику деактивации, была деактивирована и функция CheckDeactivate возвращает true, то компонента сохранит признак деактивации и больше никогда не будет обрабатывать эту логику
- Draft

Функции дочернего блюпринта

- CheckActivate - функция, которая разрешает выполнить логику активации в OnActivate
- OnActivate - функция для реализации произвольной логики на активацию этюда
- CheckDeactivate - функция, которая разрешает выполнить логику деактивации в OnDeactivate

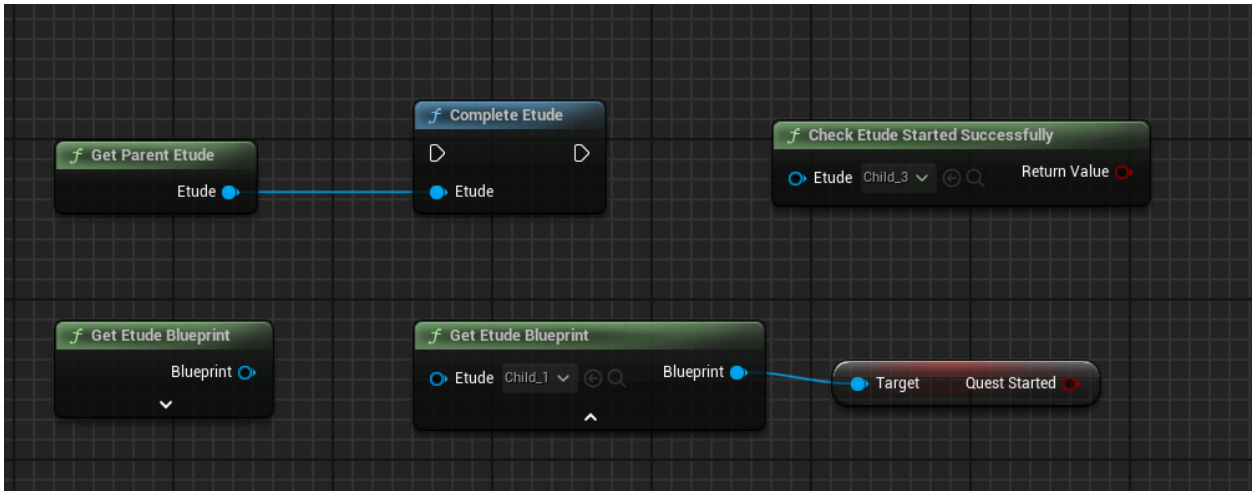
- OnDeactivate - функция для реализации произвольной логики на деактивацию этюда
- OnComplete - вызывается когда этюд завершается

Пользовательские компоненты

Используя в качестве базового класса ULogicComponent пользователь может реализовать собственные класс компонент со специфичной для конкретной игры логикой

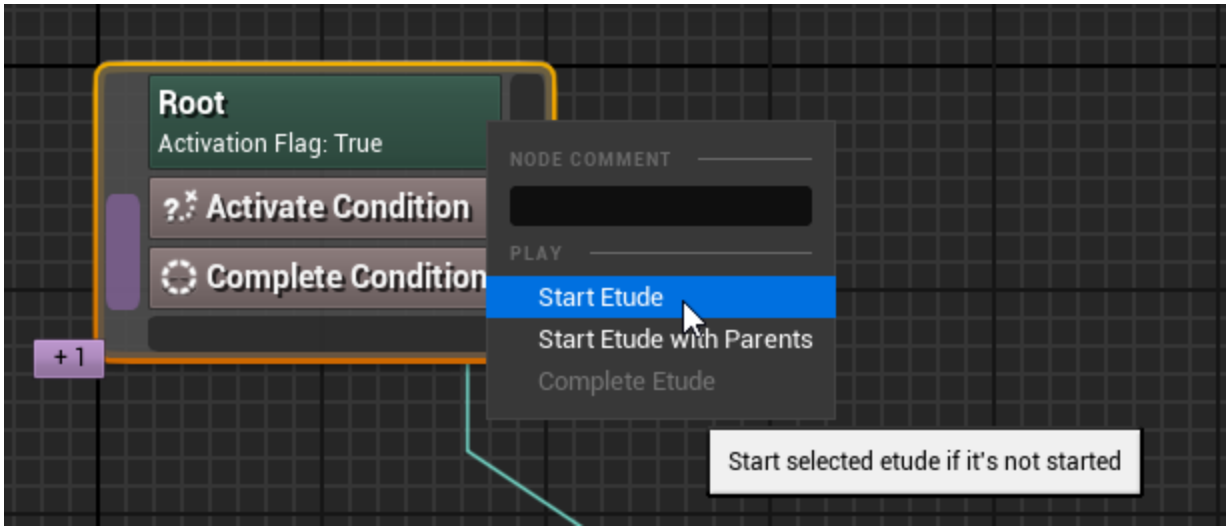
Каждая реализация класса пользовательской компоненты в проекте может расширять свое поведение специализированным классом блюпринта. Все типы компонент могут использовать расширение своего функционала путем делегирования вызовов из c++ кода в дочерний блюпринт, который автоматически создается в редакторе по требованию. Нужные функции этого класса могут помечены специальными атрибутами, которые редактор интерпретирует как пользовательские расширяемые функции и отображает в виде кнопок с разными стилями, которые также задаются через мета атрибуты

Из блюпринтов компонент можно получить доступ к родительскому этюду, его блюпринту. Менять значения их переменных, проверять состояние. Из любого блюпринта можно получить доступ к любому этюду, поскольку этюды имеют глобальную область видимости и существуют все время пока запущена игра

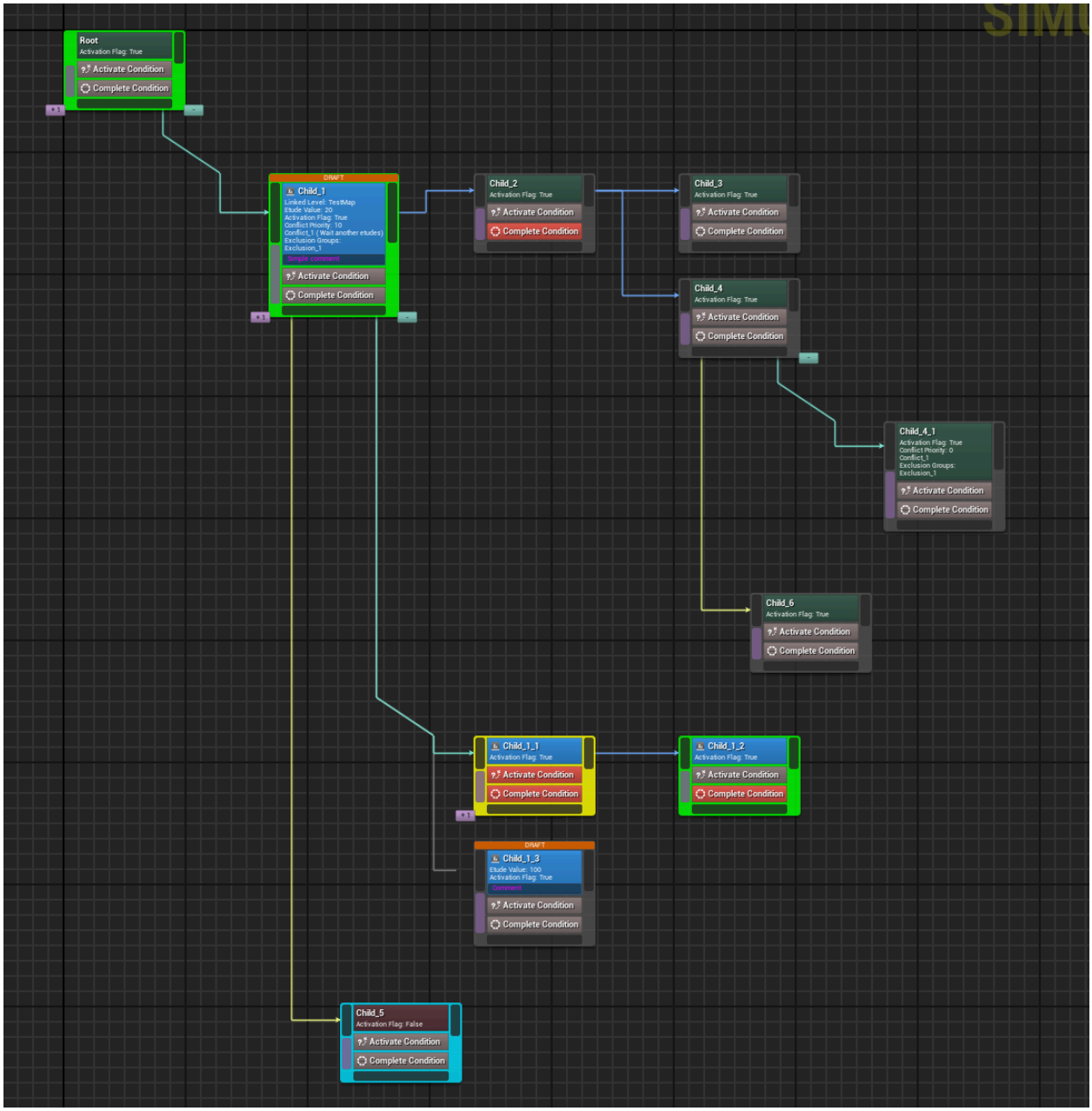


Отладка этюдов в режиме Play In Editor

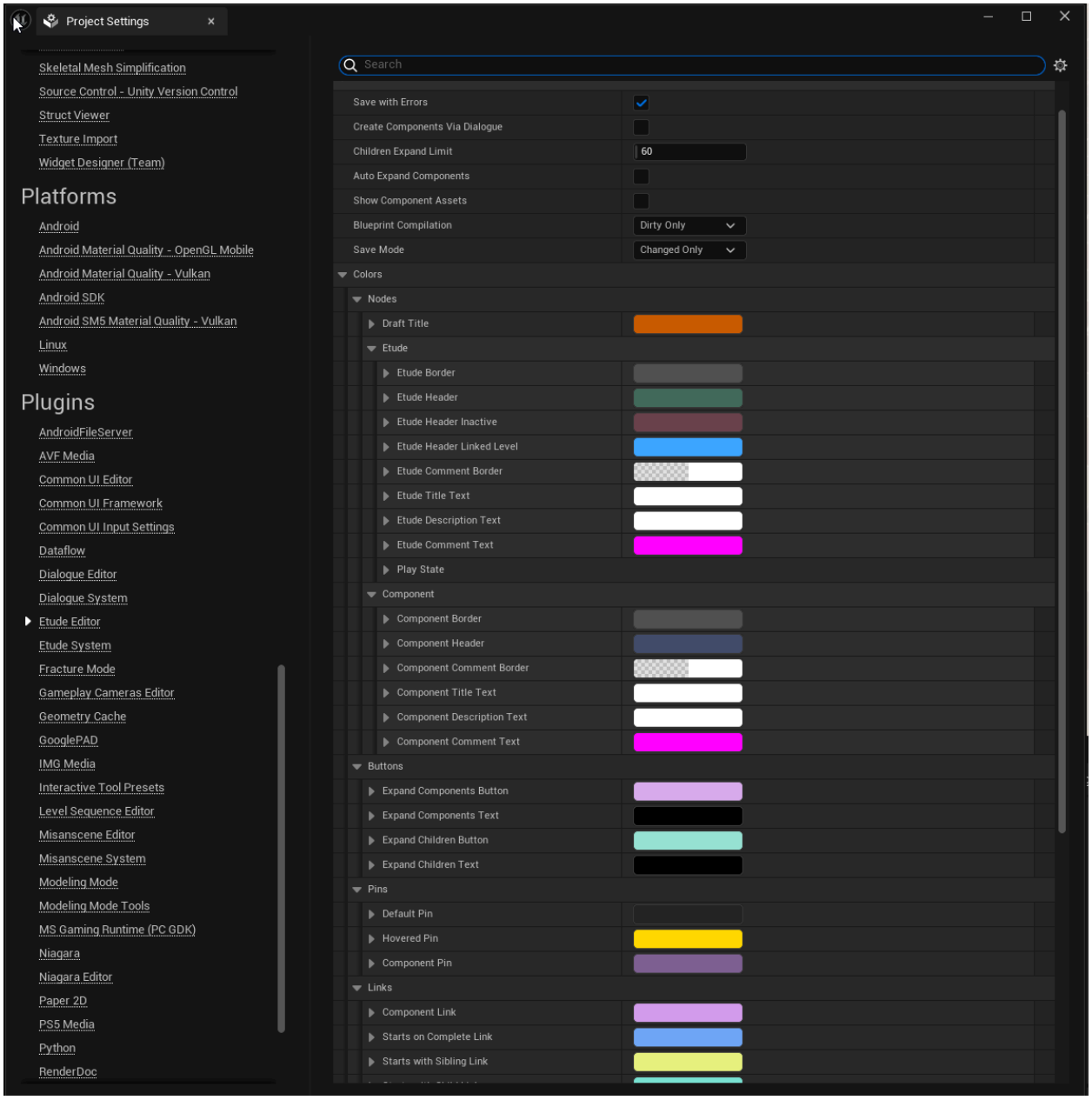
При запущенной игре в открытом редакторе этюдов можно отслеживать состояния любого этюда в графе, запускать/останавливать произвольный этюд через контекстное меню.



Можно активировать/деактивировать этюды изменяя свойство Etude Activation Flag непосредственно в панели Details. Все изменения будут отменены при завершении режима.



Все существенные параметры визуализации нод в графе вынесены в настройки и могут быть изменены пользователем. Например, цвета нод и линков между нодами в различных состояниях, отступы между нодами при автоматическом построении графа, глубина построения по умолчанию, и прочее.

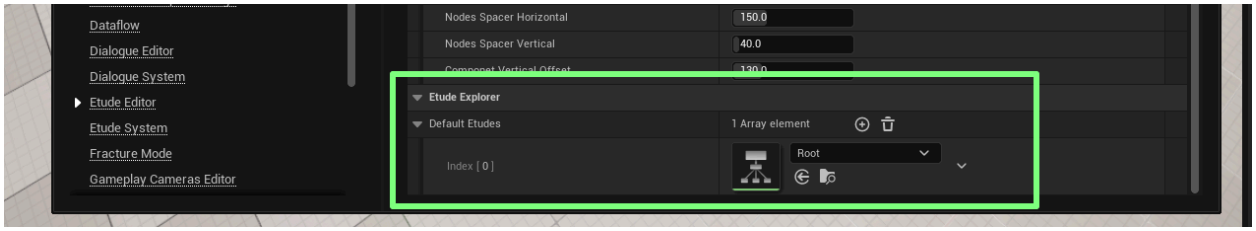


Дополнительные функции

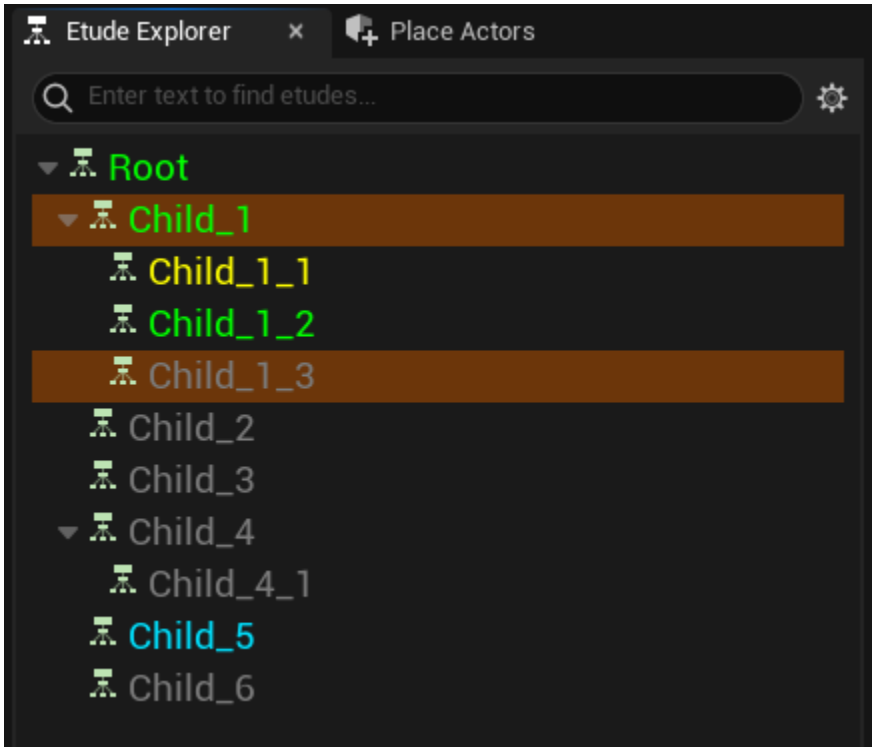
В плагине поставляется несколько вспомогательных классов, которые предоставляют дополнительный сервис в редакторе:

Etude Explorer

Стыкуемый в произвольное место виджет редактора Unreal
Открывается из меню Tools > Etude Features > Etude Explorer
Отображает всю иерархию списка этюдов, которые заданы настройках проекта

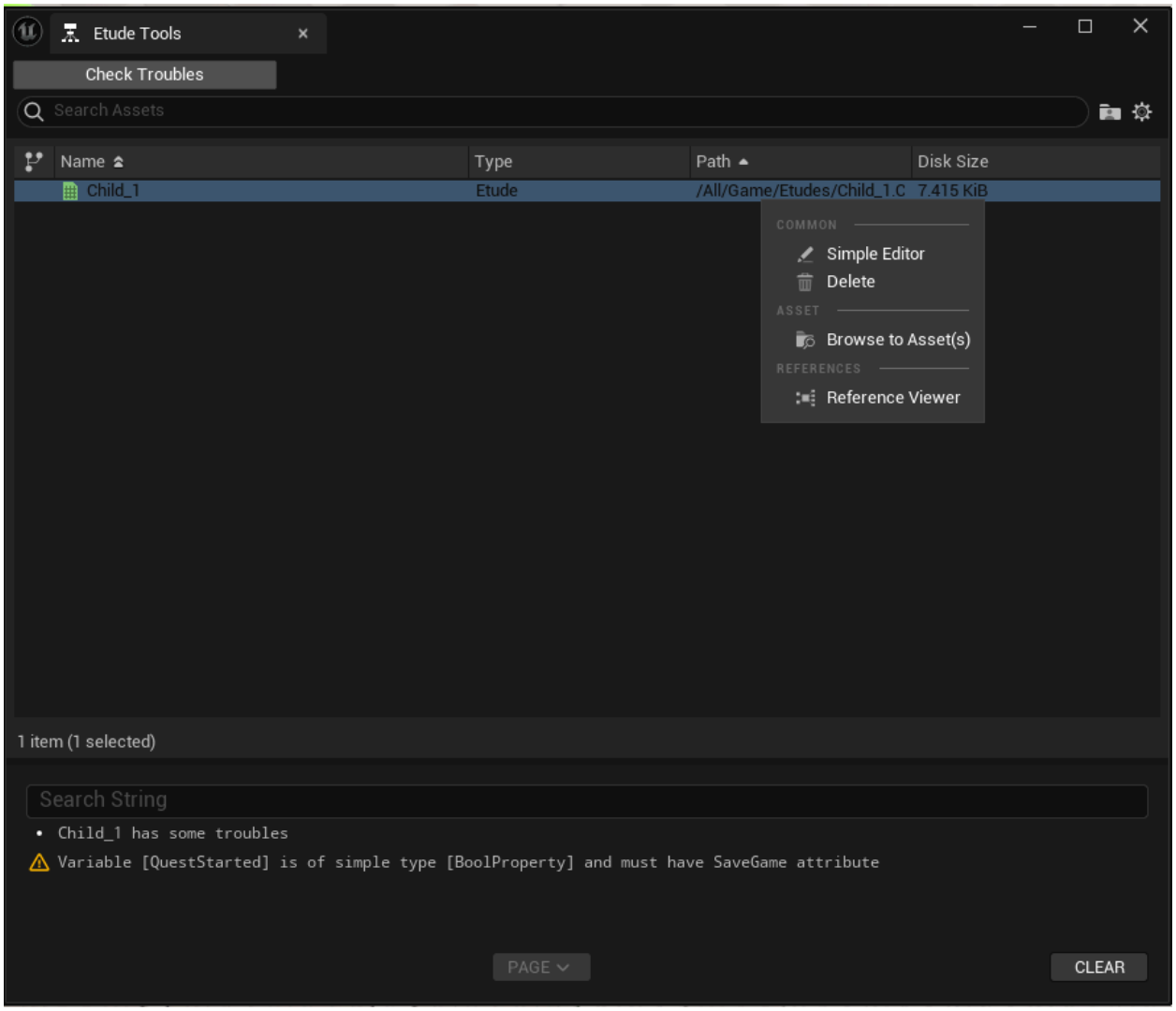


Позволяет осуществлять поиск по имени, Скрывать/показывать Draft-этюды и этюды из Developer папок. По двойному клику открывает редактор этюдов. В режиме Play In Editor отображает текущее состояние аналогично редактору этюдов.



Etude Tools

Форма для массовой проверки ассетов этюдов и компонент на целостность с визуализацией ошибок и возможностью открыть ошибочный ассет для исправления или удалить его.



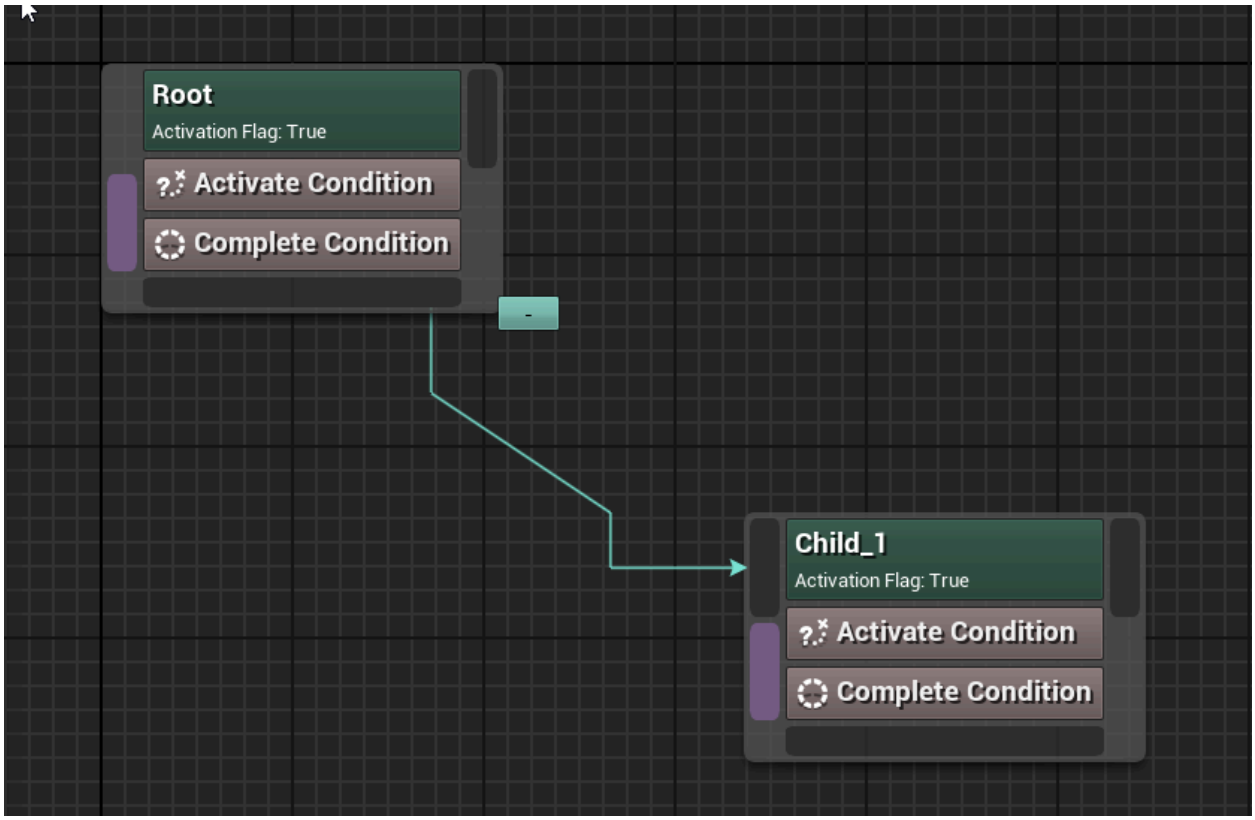
Фильтр ассетов компонент для Content Browser

Реализован классом FFrontendFilter_EtudeComponentVisibility - фильтр позволяющий управлять видимостью ассетов компонент. По умолчанию ассеты компонент скрыты

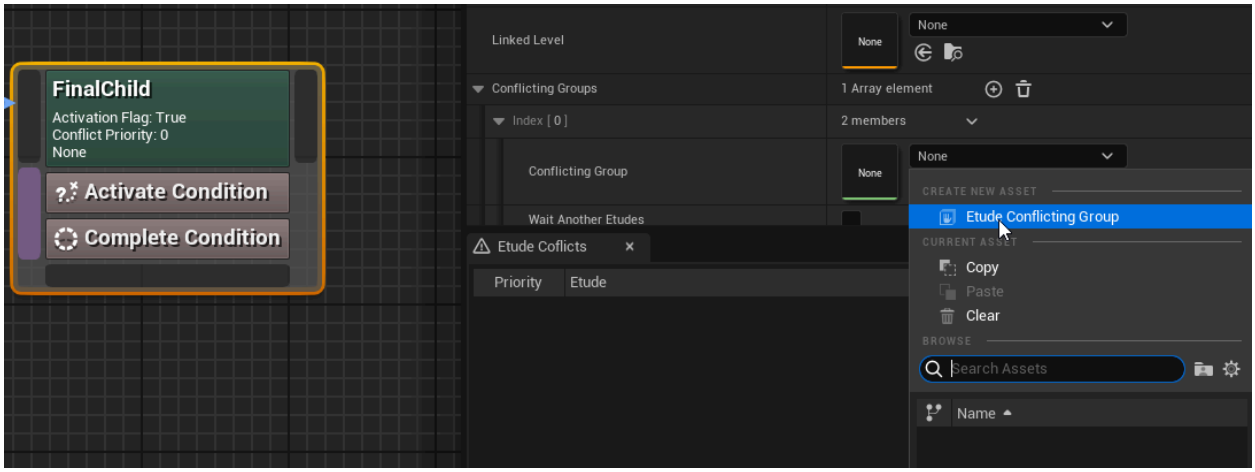
Проверка работоспособности плагина

Действия совершаются в новом пустом проекте Unreal (Empty Project - C++) после установки плагина с зависимостями.

Создание этюдов

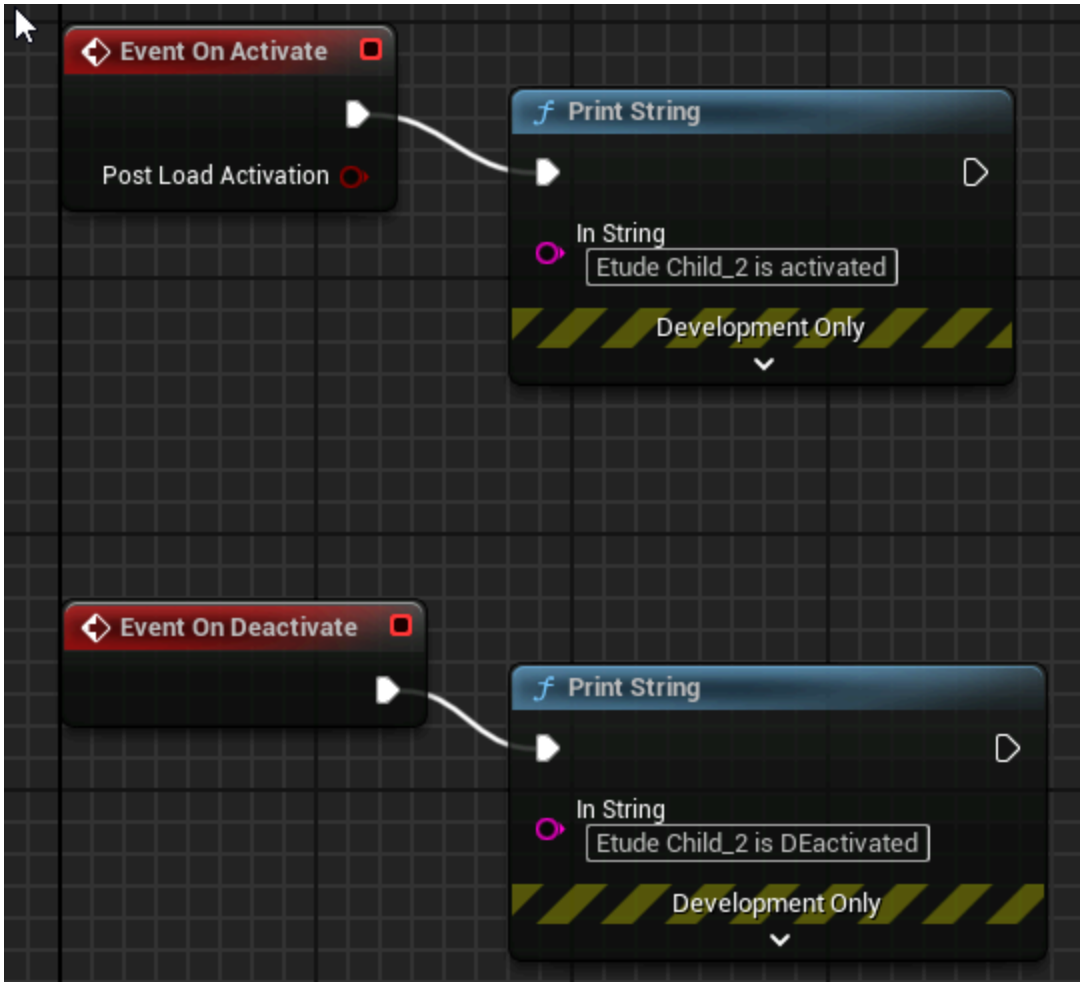
- Правый клик в окне Content Browser, выбрать Etude System/Etude
- Ввести имя нового ассета: Root
- Двойным кликом по новому ассету открыть редактор этюдов
- Зажать левую кнопку на полоске в нижней части узла Root и перетащить вниз. В открывшемся меню выбрать Etude. Ввести имя нового этюда: Child_1. Для автоматического размещения нод на графе нажмите Ctrl + Space
- 

- Повторить предыдущий пункт, создав этюд Child_2
- Перетаскиванием из *правой* части узла Child_1 создать еще один этюд, с именем FinalChild
- Кликом по узлу выбрать этюд FinalChild. В панели Details в правой части окна найти пункт Conflicting Groups, нажать на ⊕.
- Раскрыть добавленный элемент, раскрыть выпадающий список в пункте Conflicting Group, выбрать в меню Etude Conflicting Group

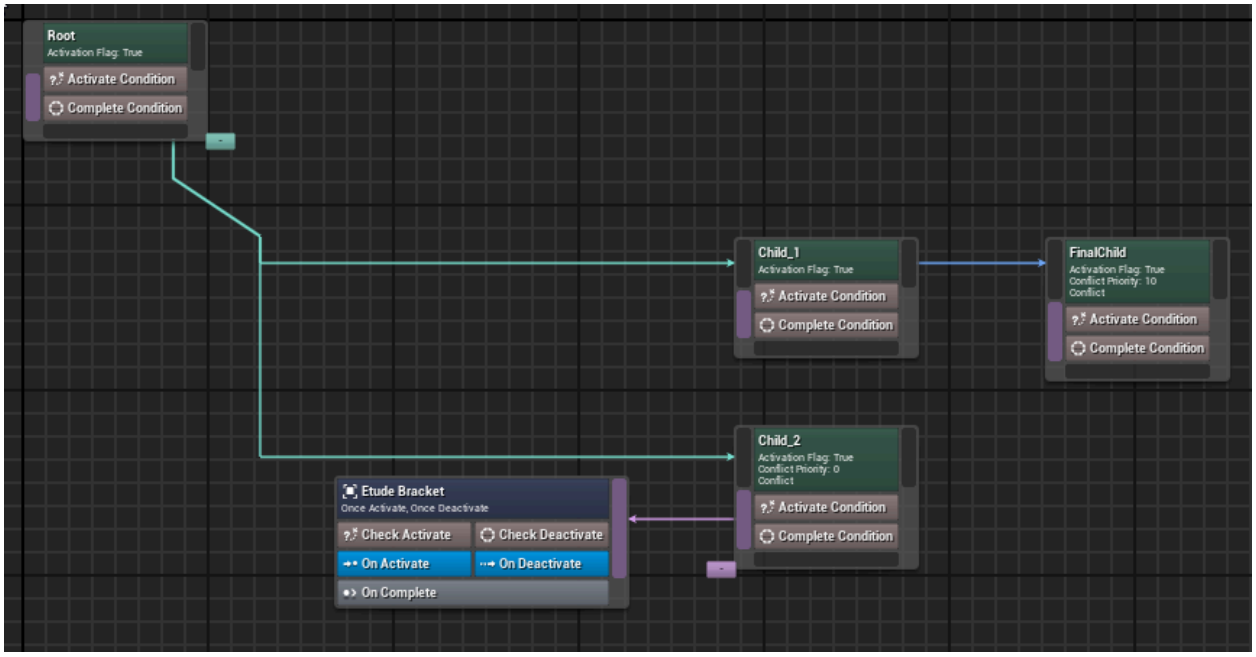
- 

- Ввести имя нового ассета: Conflict
- В строке Priority ввести 10
- Выбрать этюд Child_2. В панели Details аналогично предыдущему пункту добавить Conflicting Group. В выпадающем списке выбрать в меню существующую группу Conflict, вместо создания новой
- Перетаскиванием из левой нижней части узла Child_2 добавить новый компонент Etude Bracket
- 

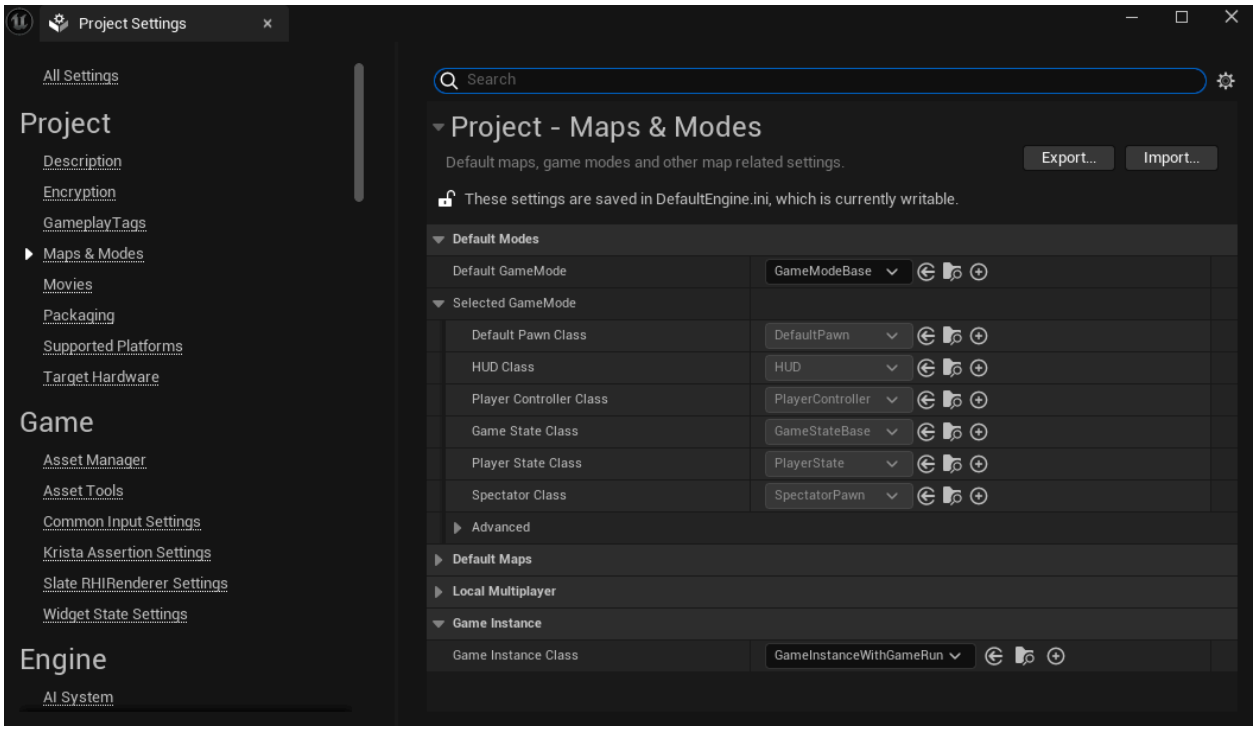
- В появившемся узле компонента сделать двойной клик по кнопке On Activate. Откроется окно блюпринта.
- Вернуться в окно редактора этюдов. Двойным кликом по кнопке On Deactivate в узле компонента создать новый ивент (снова откроется окно блюпринта)
- Собрать в этом окне следующий блюпринт:

- 

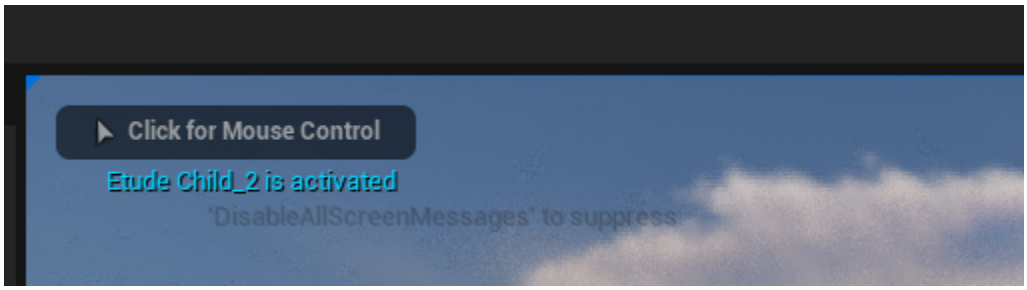
- Итоговое окно редактора этюдов должно выглядеть следующим образом (нажмите Ctrl+Space чтобы автоматически выстроить узлы в правильные места)

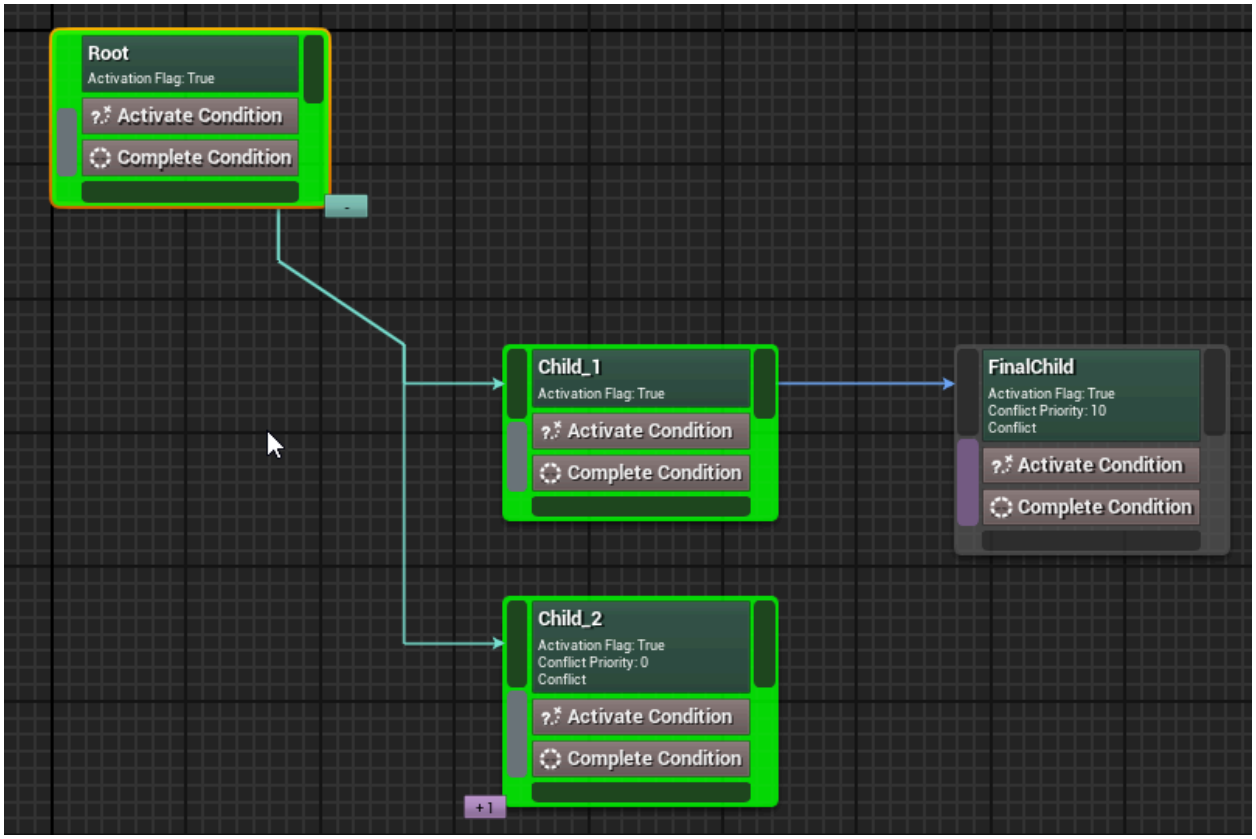
- 

Проверка этюдов в игре

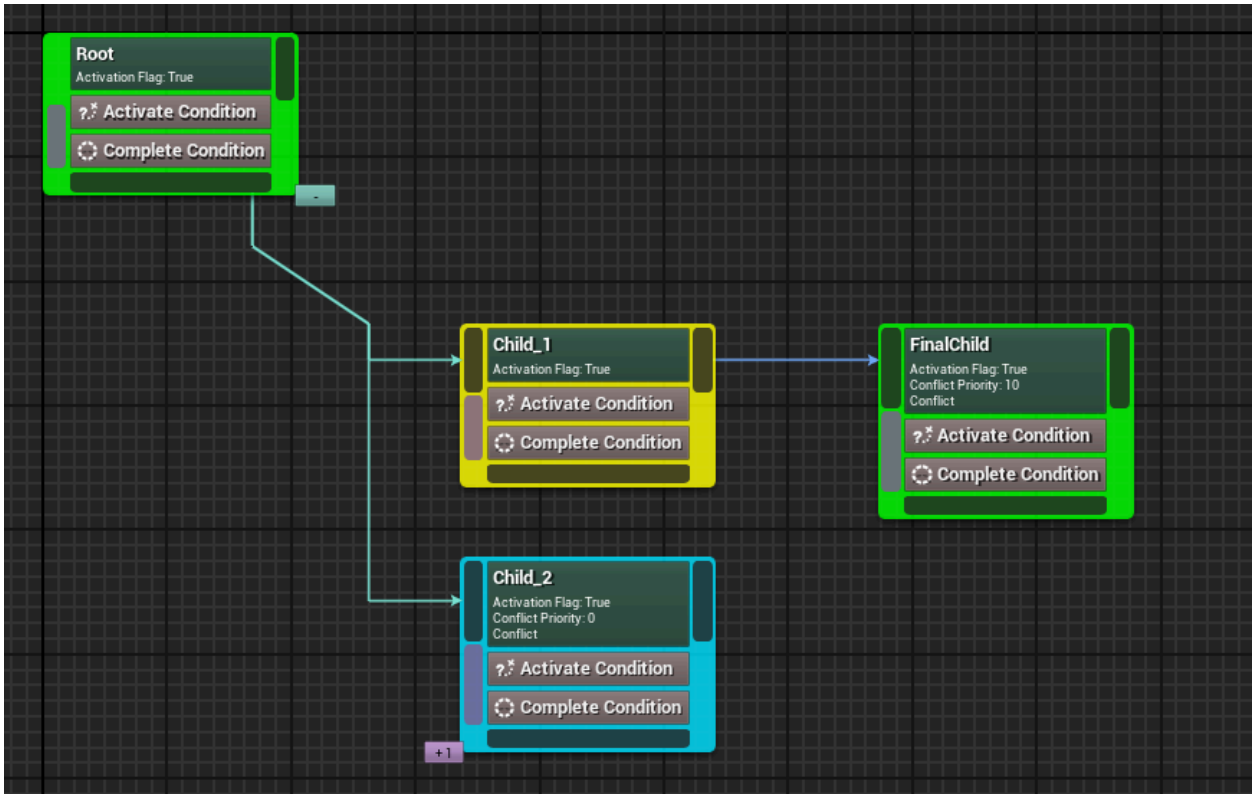
- Открыть меню Edit/Project Settings/Maps & Modes и выбрать в качестве GameInstance класс GameInstanceWithGameRun
- 

- Запустить игру

- Открыть редактор этюдов (переместите его окно так, чтобы видеть одновременно и окно с запущенной игрой), нажать правой кнопкой на узел Root и выбрать в меню Start Etude
- Удостовериться, что на экране и в логе появилась надпись Etude Child_2 is activated
- 

- В редакторе этюдов активные этюды отмечаются зеленым цветом:
- 

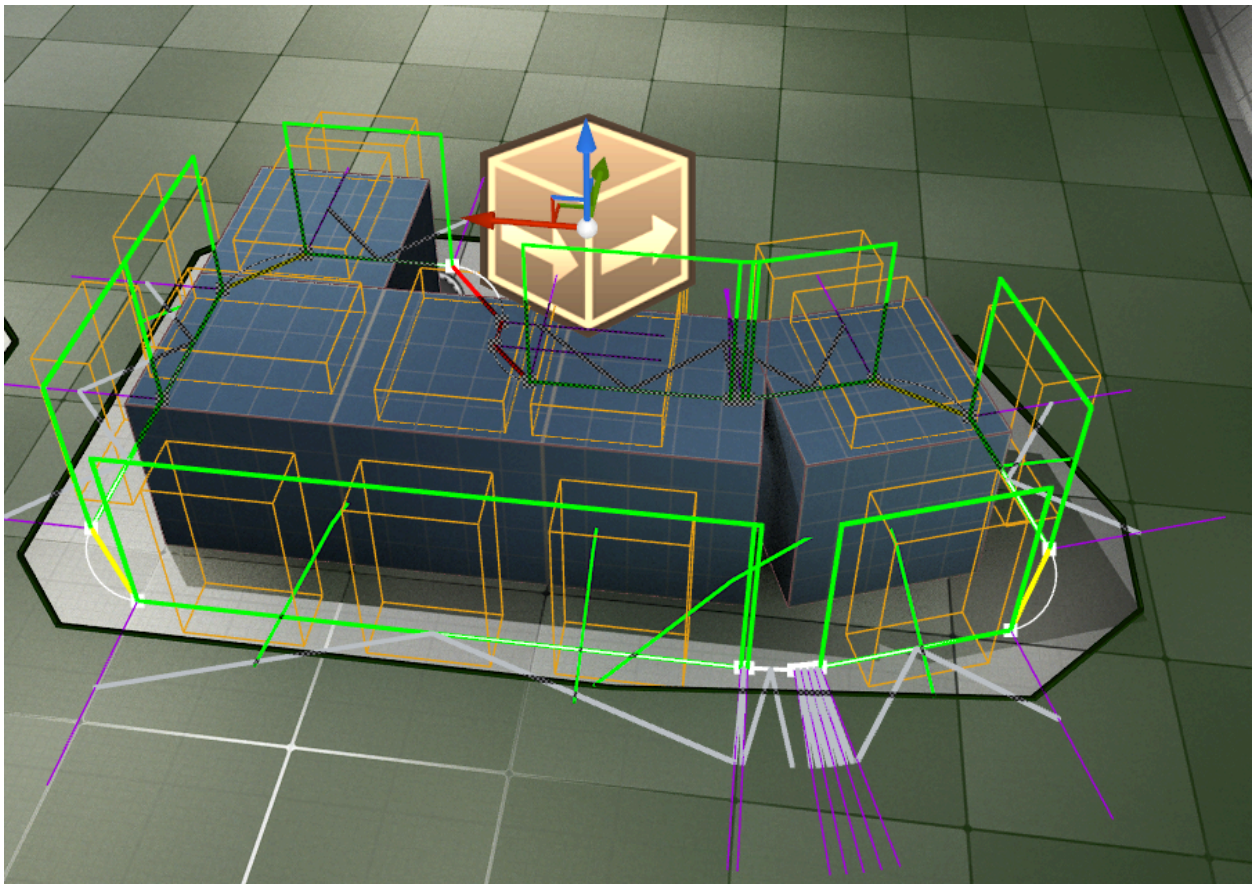
- Нажать правой кнопкой на узел Child_1 и в контекстном меню выбрать Complete Etude
- Удостовериться, что на экране и в логе появилась надпись Etude Child_2 is DEactivated
- В редакторе этюдов завершенный этюд помечается желтым цветом, а неактивный из-за конфликта - голубым

- 
Root
Activation Flag: True
Activate Condition
Complete Condition
-
Child_1
Activation Flag: True
Activate Condition
Complete Condition
FinalChild
Activation Flag: True
Conflict Priority: 10
Conflict
Activate Condition
Complete Condition
Child_2
Activation Flag: True
Conflict Priority: 0
Conflict
Activate Condition
Complete Condition
+1

Cover system plugin

Зачем нужен cover system plugin

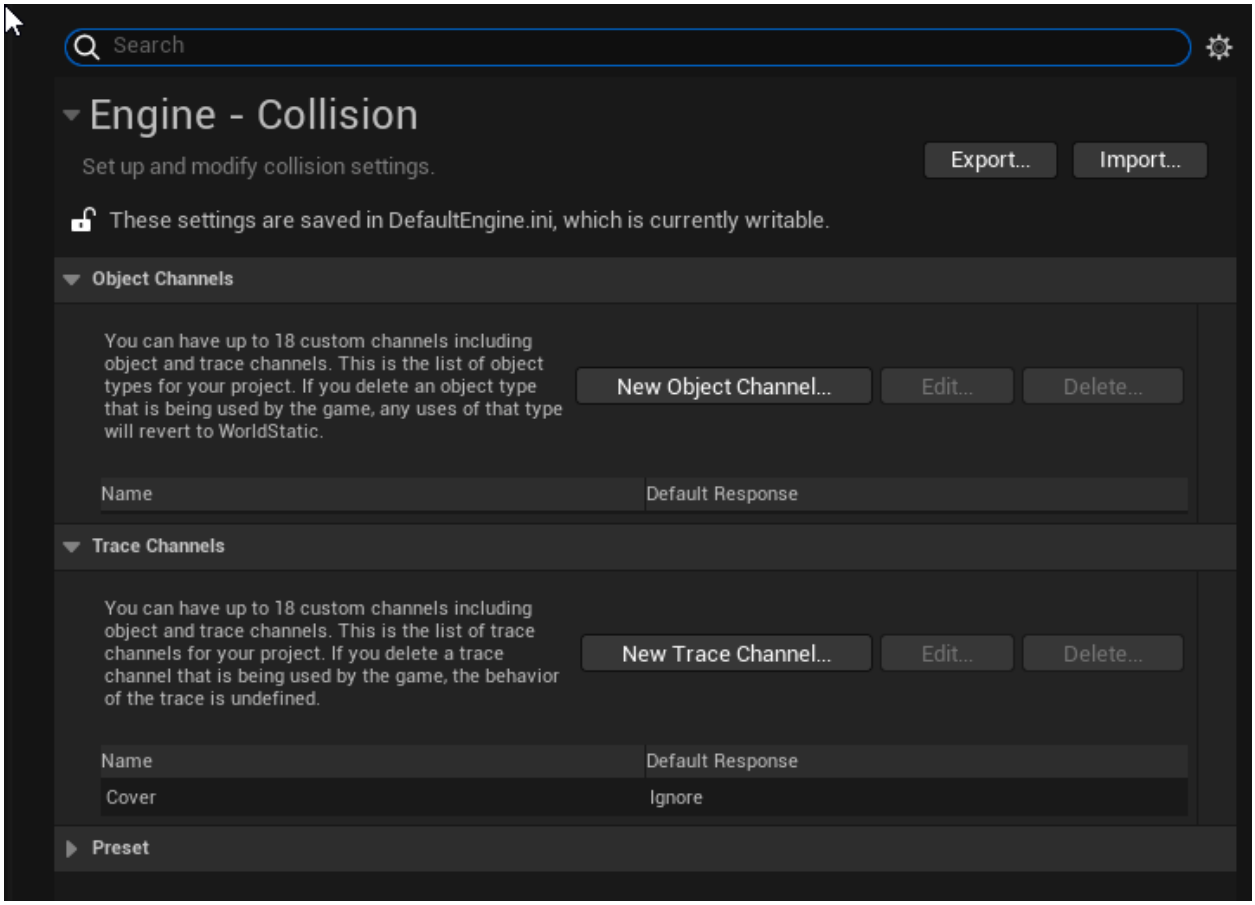
Cover system - плагин для Unreal Engine 5, генерирующий специальную разметку вокруг объектов, помеченных как укрытия. Разметка состоит из сплайна, на котором отмечены проходимые, непроходимые и особые сегменты, объектов cover slot которые может использовать игрок или искуственный интеллект для резервирования места за укрытием и cover surface для разметки поверхностей укрытия, на которые можно ставить предметы.



Как установить плагин в проект

1. Установить плагины KristaMisc, KristaAssertions и GameRun по их инструкции.
2. Перенести директорию CoverSystem в директорию Plugins проекта.
3. Запустить редактор

4. Перейти в меню Edit -> Plugins. Найти плагин CoverSystem и поставить галку напротив. Закрыть редактор
5. Перейти в меню Edit -> Project Settings. Найти в секции Collision пункт Trace Channels и создать там новый канал для укрытий с именем Cover и ответом по умолчанию Ignore

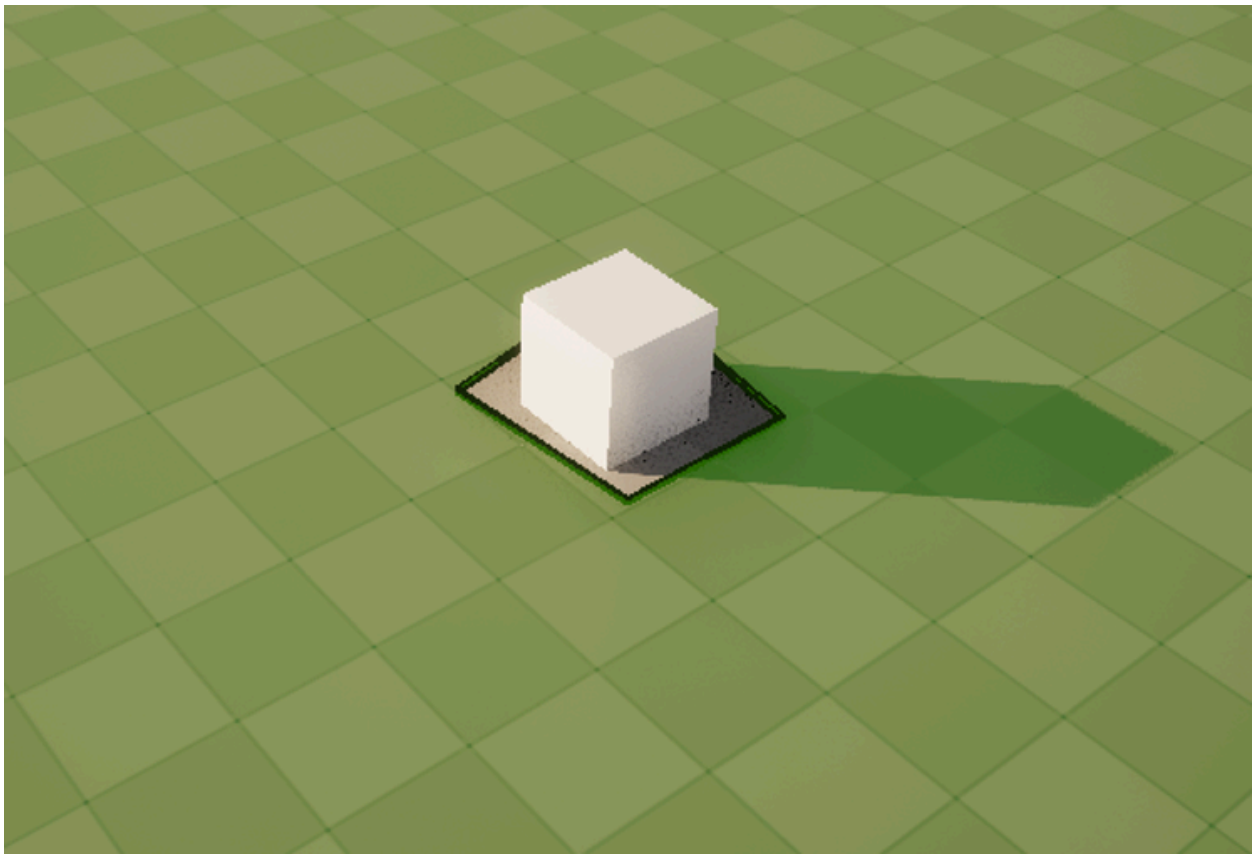


6. Найти в секции Cover System раздел Collision и указать там созданный канал Cover в пункте Cover Query Channel
7. В той же секции поставить галку Generation - Generate Automatically
8. Теперь при создании и перемещении акторов укрытия будут создаваться автоматически.

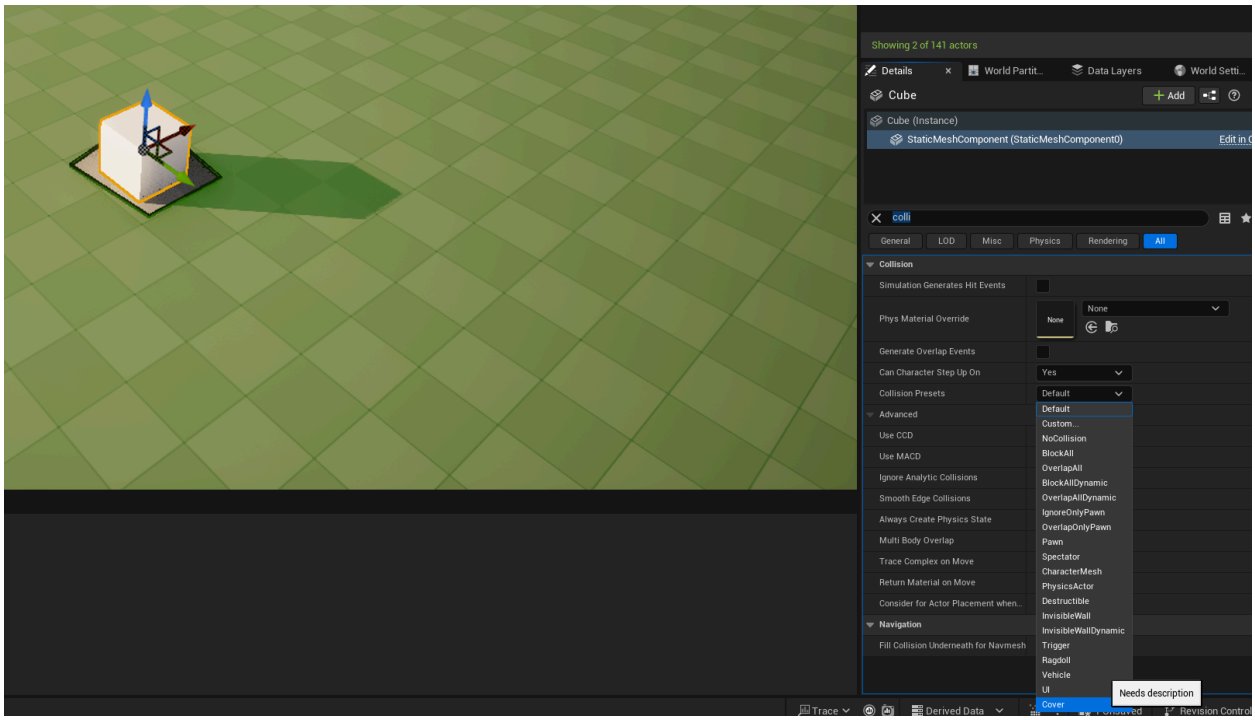
Использование плагина

1. При помощи Place Actors поставьте на уровень Nav Mesh Bounds Volume. Убедитесь что в месте, где вы хотите поставить кавер, есть навмеш (включив режим отображения навигации нажатием клавиши P)

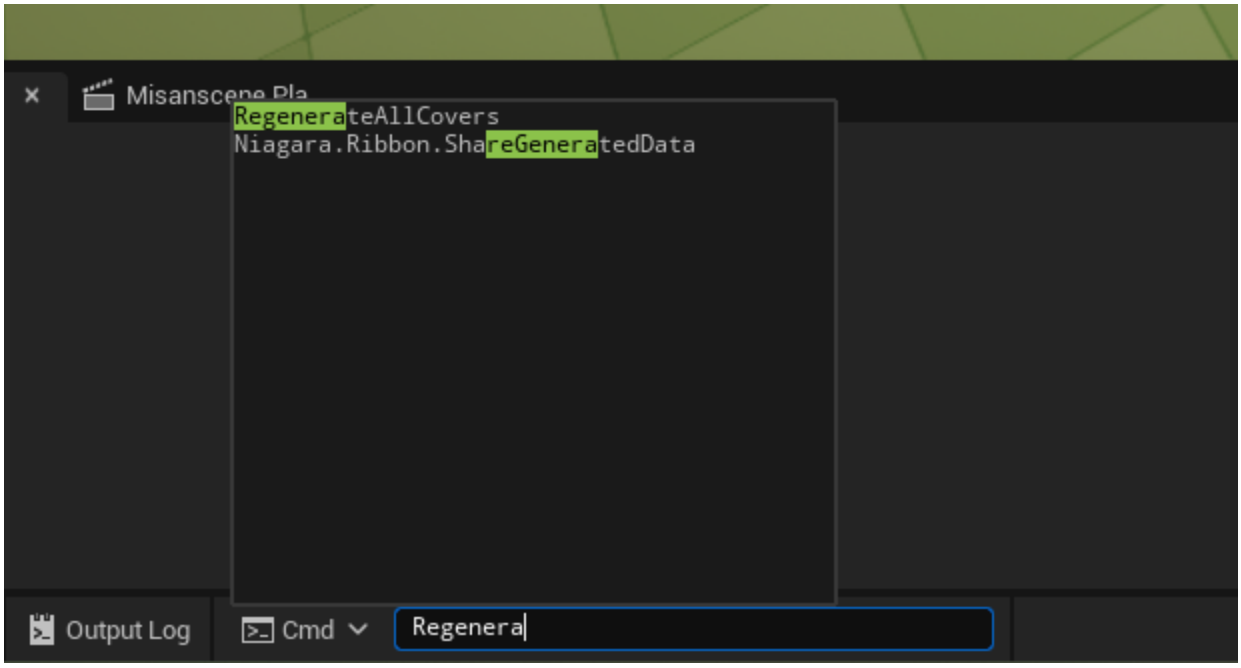
2. Добавьте на уровень примитив. Например Box Shape. Разместите его на полу где есть навмеш.



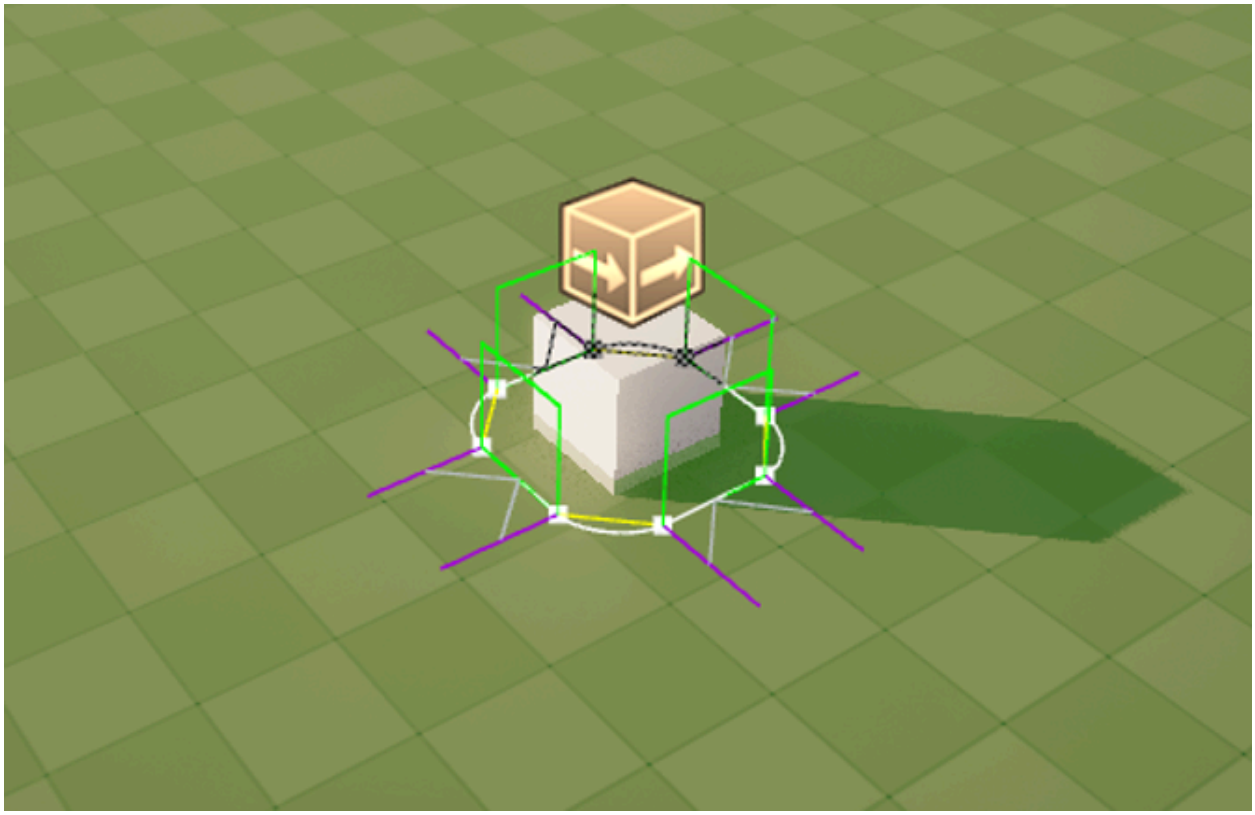
3. Переключите коллизии на пресет, в котором блокируется канал Cover, или поставьте коллизии Custom и вручную выставьте канал Cover в Block.



4. Подвигайте примитив (плагин реагирует на изменения) или напишите в консоль редактора RegenerateAllCovers



5. Кликните на появившуюся иконку укрытия и посмотрите на результат



Как работает плагин

После подключения и настройки плагина, он найдет все primitive компоненты с коллизиями, указанными в настройках, сгруппирует эти компоненты в физические укрытия (то есть несколько static mesh компонентов, лежащих рядом будут определены как единое укрытие), после чего для каждого укрытия создаст дополнительный актор, содержащий информацию об этом укрытии и сплайн вокруг него.
Все изменения укрытий будут автоматически запрашивать перегенерацию, но можно вызвать ее вручную, если нужно консольной командой RegenerateAllCovers, или использовать CallInEditor метод RegenerateCoverData у актора ACoverData
Обратите внимание, что триггер длля перегенерации данных - изменение primitive компонентов, так что если вы настроили плагин после создания уровня, то нужно вызвать первую генерацию вручную.
Кроме того плагин позволяет заранее рассчитать какие укрытия из каких позиций видны. Результаты расчетов сохраняются в UCoverSlotComponent. На этапе редактора можно произвести предварительные расчеты видимостей. В дальнейшем во время запуска игры если произошли какие-то изменения на локации, в системе UCoverRegistrySubsystem можно запросить обновить данные по видимостям между укрытиями. Пересчет видимости выполняется в потоке физики, за обработку отвечает класс FCoverVisiblityAsyncInput.

Как понять что плагин работает

Если плагин создал ACoverData, то возле физического кавера появится спрайт, сигнализирующий о том, что сплайн построен. На спрайт можно кликнуть и активируется дебажная отрисовка сплайна, как на скриншоте

Настройка

Гланая настройка, которую нужно произвести - CoverQueryChannel. В ней нужно указать канал коллизий, который отличает укрытия от других primitive объектов. Остальные настройки вторичны и нужны лишь для установки метрик под ваши нужды

Ключевые классы плагина

ACoverData

Актер для информации о кавере. Нужен как массив компонентов. Есть CallInEditor функция для принудительной перегенерации кавера

UCoverRegistrySubsystem

Подсистема, содержащая соответствие между физическими укрытиями и ACoverData

UCoverSystem_Settings

Список настроек плагина, используемый всеми остальными классами

UCoverSplineComponent

Наследник USplineComponent, данный сплайн строится вокруг укрытий с указанным в настройках каналом коллизий. Сплайн всегда замкнутый, отступает от укрытия на велечину, заданную в настройках. Хранит всю важную информацию в переопределенных метаданных

UCoverSplineMetadata

Содержит игровую информацию о сплайне. Каждый участок сплайна(от точки i до i+1) может быть проходимым или непроходимым. Проходимые участки делятся на обычные проходимые, переходные(повороты за угол), безусловные переходные(повороты за малые углы). Данную информацию можно использовать для создания локомоушена

UCoverSlotComponent

Слот на проходимом участке сплайна. Содержит информацию о том, какие действия можно совершить на данном участке сплайна(например можно ли высовываться из-за него, и если да, то с какой стороны), а так же содержит логику резервации слота.

UCoverSurfaceComponenet

Работает аналогично UCoverSlotComponent, но располагается не на проходимых участках сплайна, а сверху укрытия. Для случаев, если на укрытие нужно что-то поставить(например пулемет).

UCoverSystemEditorSubsystem

Система для редактора, которая содержит функции для обновления каверов. Функции доступны для запуска через консольные команды в проекте, к которому подключен данный плагин.
Также в данной системе выполняется запуск для расчета видимости каверов. Данные о видимости сохраняются в каждый UCoverSlotComponent.

Общий обзор

Система предназначена для генерации скелетных мешей (моделей голов) путём смешения набора пресетных голов (каждая из которых имеет собственный Skeletal Mesh и соответствующий DNA-файл), а также для последующего применения преобразований на основе данных DNA. На выходе формируется сгенерированный скелетный меш, который может использоваться в редакторе или на уровне рантайма. Кроме того, система поддерживает добавление аксессуаров (очки, наушники и т.д.), которые могут быть автоматически "подогнаны" под полученную меш-модель.

Основные составляющие системы:

1. **UHeadGeneratorAsset** – Data Asset, содержащий базовую информацию о том, какие меши смешиваются, а также логика инициализации и перегенерации данных.
2. **UHeadGeneratorComponent** – Компонент актора, который взаимодействует с UHeadGeneratorAsset, создаёт результирующие меши, применяет веса смешения и управляет аксессуарами.
3. **FDNAGenerator** – Структура для работы с DNA: загрузка DNA, маппинг вершин и костей, применение изменений к целевому мешу на основе DNA.
4. **FSkeletalMeshGenerator** – Структура для генерации финального скелетного меша из базового меша и набора целевых мешей (пресетов) с использованием заданных весов смешения.

UHeadGeneratorAsset

Класс: `UHeadGeneratorAsset : public UDataAsset`

Назначение:
Хранит ссылки на базовый скелетный меш, структуру для генерации мешей и для генерации DNA, а также вспомогательные данные. Предполагается, что на основе набора пресетов (BlendPartsAsset) система может "спечь" (Bake) результаты в итоговый меш.

Основные члены:

- `BaseMesh`: Исходный (базовый) скелетный меш.
- `BlendPartsAsset`: Набор данных о смешиваемых частях головы (предположительно ссылка на другой UDataAsset, где перечислены пресеты).
- `MeshGenerator`: Экземпляр `FSkeletalMeshGenerator`, выполняющий смешивание геометрии.
- `DNAGenerator`: Экземпляр `FDNAGenerator` для работы с данными DNA.

Основные методы:

- `BakeParts()`: Вызывается из редактора. Перегенерирует части головы и сохраняет результат.
- `WasInit()`: Проверяет, была ли уже выполнена инициализация.
- `Initialize(bool bForce)`: Инициализирует генератор мешей и DNA (вызывается принудительно, если `bForce == true`).
- `CreateWeightsArray(const TMap<FGameplayTag, float>& MainWeights, const TMap<FGameplayTag, float>& CorrectionWeights)`: Создаёт массив весов смешения исходя из переданных наборов тегов и значений.

UHeadGeneratorComponent

Класс: `UHeadGeneratorComponent : public USkeletalMeshComponent`

Назначение:
Компонент, отвечающий за непосредственное применение сгенерированных мешей к SkeletalMeshComponent, включая генерацию итогового меша, применение весов смешения, подключение аксессуаров. Предполагается, что данный компонент размещается на акторе, чтобы на лету (или в редакторе) менять внешний вид головы.

Основные члены:

- `GeneratorAsset`: Ссылка на `UHeadGeneratorAsset`, откуда берутся данные о базовом меше и правила смешения.

Основные методы:

- `CreateGeneratedMesh(bool bForce)`: Создаёт (или пересоздаёт) итоговый меш на основе `GeneratorAsset`. Параметр `bForce` позволяет принудительно обновить данные, даже если они были сгенерированы ранее.
- `ApplyBlendWeights(TArray<float> Weights)`: Применяет к текущему мешу указанный массив весов смешения, вызывая логику из `FSkeletalMeshGenerator`.
- `EnableAccessories(TArray<TSoftObjectPtr<USkeletalMesh>> Accessories)`: Активирует набор аксессуаров. Каждый аксессуар клонируется и "вешается" на итоговый меш.
- `EnableAccessoriesByIndex(TArray<int32> Accessories)`: Аналогично, но по индексам, которые предполагается брать из какой-то внутренней логики или массива.

- `UpdateAccessory(int32 Index, const TArray<FTransform>& BoneGlobalTransforms)`: Обновляет аксессуар при изменении положения костей или при обновлённом меше.
- `GetAccessory(int32 Index)`: Возвращает аксессуар по индексу.
- `GetActiveAccessories()`: Возвращает все активные аксессуары.
- `GetActiveAccessoryBySource(TSoftObjectPtr<USkeletalMesh> SourceAccessory)`: Возвращает активный аксессуар по исходному референсу меша.

Примечания:
Основная задача — применение итоговой головы к SkeletalMeshComponent, а также управление аксессуарами (подключение/отключение, обновление).

FDNAGenerator

Структура: `FDNAGenerator`

Назначение:
Занимается загрузкой и применением данных DNA к скелетному мешу. DNA файлы содержат информацию о позициях вершин, пропорциях, костях и прочих параметрах морфинга лицевой анимации.

Основные члены:

- `DNAFileName`: Имя файла DNA для загрузки.
- `DNAReaderAxisX`, `DNAReaderAxisY`, `DNAReaderAxisZ`: Настройки координатной системы для интерпретации данных DNA.
- `IndexMap`: `FSkelMeshToDnaIndexMap` для соответствий между индексами в DNA и SkeletalMesh.
- `DNADataArray`: Прочитанные байты DNA.
- `CachedDNAFileName`, `CachedBaseMesh`: Кэшированные значения, чтобы не переинициализировать без необходимости.
- `bWasInitialized`: Флаг инициализации.

Основные методы:

- `bool Init(const USkeletalMesh* BaseMesh, bool bForce)`: Инициализирует генератор на основе базового меша. Может подгружать DNA из файла или брать из меша, если DNA уже встроено.

- `bool UpdateDNA(USkeletalMesh* TargetMesh, const TArray<FMeshLODData>& MeshLODData, const TArray<FTransform>& BoneGlobalTransforms)`: Применяет данные DNA к целевому мешу, изменяя позиции вершин или положения костей.
- `bool HasDNA() const`: Проверяет, успешно ли загружено DNA.
- `bool WasInitializedWith(const USkeletalMesh* BaseMesh) const`: Проверяет, совпадает ли указанный базовый меш с тем, для которого уже была инициализация.

FSkelMeshToDnaIndexMap

Структура: `FSkelMeshToDnaIndexMap`

Назначение:
Хранит и вычисляет соответствия между индексами вершин и костей в DNA и теми же сущностями в SkeletalMesh. Когда DNA применяет изменения, нужно знать, к каким вершинам и костям в результирующем меше они относятся.

Основные методы и поля:

- `InitJointMapping(FDNAAdapter& Adapter, const USkeletalMesh* SkelMesh)`: Инициализирует маппинг костей.
- `InitVertexMapping(FDNAAdapter& Adapter, const USkeletalMesh* SkelMesh)`: Инициализирует маппинг вершин.
- `GetMapped(int32 DNAMeshIndex, int32 DNAVertexIndex)`: Возвращает, какой именно индекс вершины в рендер-данных меша соответствует индексам из DNA.
- `GetMapped(int32 DNAJointIndex)`: Возвращает индекс кости в меше для данной кости DNA.
- `bWasJointMapBuilt`, `bWasVertexMapBuilt`: Флаги успешной инициализации маппинга.

Структура участвует при применении DNA к финальному мешу, чтобы правильно сдвигать вершины и кости.

FSkeletalMeshGenerator

Структура: `FSkeletalMeshGenerator`

Назначение:
Осуществляет процедуру смешения набора скелетных мешей (пресетов) с базовым мешем по указанным весам, формируя итоговый меш. Это ядро геометрической части: создает интерполированный набор вершин и костей на основе нескольких источников.

Основные поля:

- `CachedBaseMesh`: Кэшированный базовый меш.
- `CachedTargets`: Массив целевых мешей (пресетов), которые будут смешиваться.
- `BaseMeshData`: Данные о базовом меше в удобном формате (LODы, вершины, тангенты и т.д.).
- `BaseMeshRawBonePose`: Исходный поз костей базового меша.
- `BlendShapes`: Набор BlendShape данных, получаемый после анализа Target мешей.
- `bIgnoreCache`: Игнорировать ли кэш (перегенерировать всё заново при следующем запросе).
- `PositionThreshold`, `TangentThreshold`: Пороговые значения для определения, какие вершины считать "одинаковыми" (важно для оптимизации и точности смешивания).

Основные методы:

- `bool Init(const USkeletalMesh* Base, const TArray<const USkeletalMesh*>& Targets, bool bForce)`: Инициализирует генератор, подготавливая `BaseMeshData`, `BlendShapes` и т.д.
- `bool ApplyToMesh(USkeletalMesh* Mesh, const TArray<float>& Weights, TArray<FMeshLODData>& MeshFinalData, TArray<FTransform>& MeshBonesGlobal) const`: Применяет набор весов к базовому мешу и пресетам, генерируя финальные вершины и кости.
- `int GetBlendShapesCount() const`: Возвращает количество BlendShape вариантов, загруженных в генератор.
- `bool WasInit() const`: Проверяет, инициализирован ли генератор.

Примечания по работе:

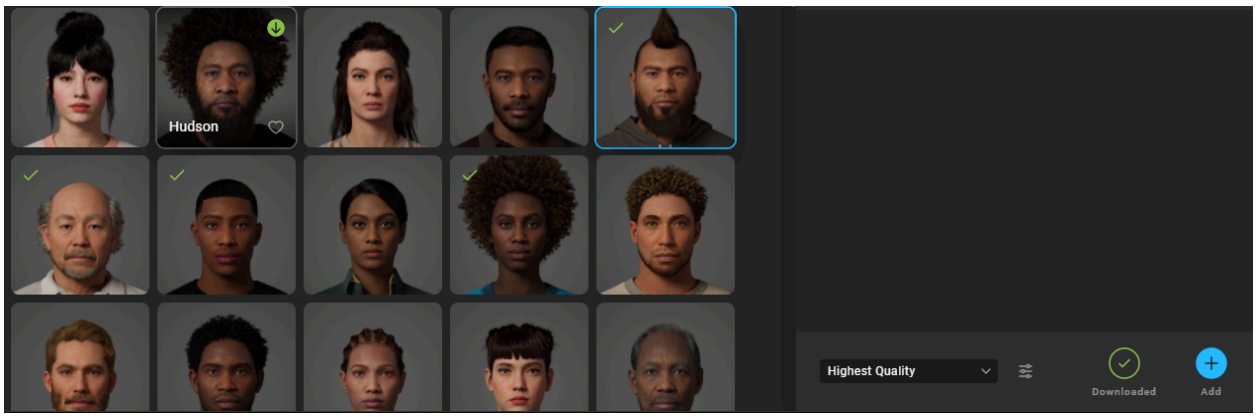
- При смешении каждая вершина нового меша вычисляется как взвешенная сумма соответствующих вершин в `BaseMesh` и `Targets`.
- Аналогично кости могут подстраиваться, если это предусмотрено BlendShape’ами.
- `RebuildBlendShapesImportedData()` — приватный метод, который выполняет основную работу по извлечению BlendShape данных из `Base` и `Targets`.

Общий сценарий использования

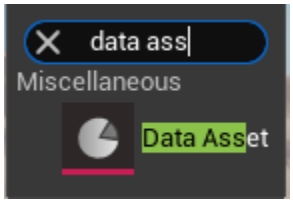
1. **Подготовка UHeadGeneratorAsset**:
    a. Создайте и настройте `UHeadGeneratorAsset`.
    b. Добавьте в него набор скелетных мешей голов (пресетов), которые будут смешиваться. Один из мешей должен быть помечен как базовый и иметь соответствующий ему DNA-файл.
    c. При инициализации ассет самостоятельно настроит:
        i. `FSkeletalMeshGenerator` для интерполяции геометрии мешей.
        ii. `FDNAGenerator` для применения данных DNA.
2. **Использование компонента (UHeadGeneratorComponent)**:
    a. Создайте на акторе компонент типа `UHeadGeneratorComponent` и укажите ему созданный `UHeadGeneratorAsset`.
    b. Вызовите `ApplyBlendWeights(TArray<float> Weights)` для применения весов смешения голов. Внутри:
        i. Генерируется итоговый меш, смешивающий все пресеты в соответствии с указанными весами.
        ii. К мешу применяются данные DNA, корректируя вершинные позиции и костные трансформации.
    c. В результате вы получите готовую сгенерированную голову.
3. **Добавление аксессуаров**:
    a. Если вам необходимо добавить различные аксессуары (очки, наушники и т.д.), используйте методы компонента:
        i. `EnableAccessories(TArray<TSoftObjectPtr<USkeletalMesh> > Accessories)` – позволяет загрузить и активировать указанный набор аксессуаров.
        ii. Или `EnableAccessoriesByIndex(TArray<int32> Accessories)` – если у вас есть индексы аксессуаров, привязанные к логике выбора.
    b. После добавления аксессуаров, система автоматически клонирует их меши, при необходимости обновляет их положение относительно костей сгенерированной головы (через `UpdateAccessory(...)` вызовы) и применяет их к текущему скелетному мешу.
4. **Гибкость системы**:
    Если вы не хотите использовать `UHeadGeneratorComponent`, вы можете руками воспроизвести его функционал:
    a. Вызывайте методы `UHeadGeneratorAsset` для генерации результирующего меша и применения DNA.
    b. Работайте с аксессуарами, клонируя и позиционируя их относительно итогового меша вручную, используя соответствующие функции обновления.

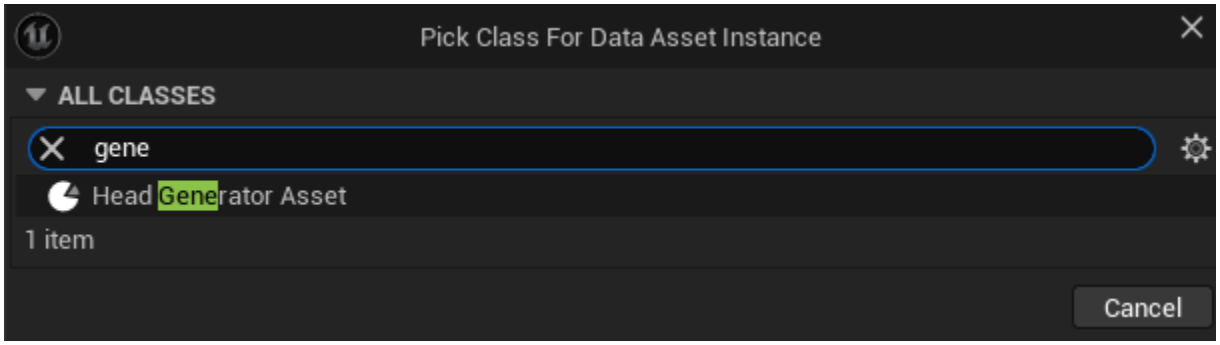
Таким образом, общий подход выглядит так:

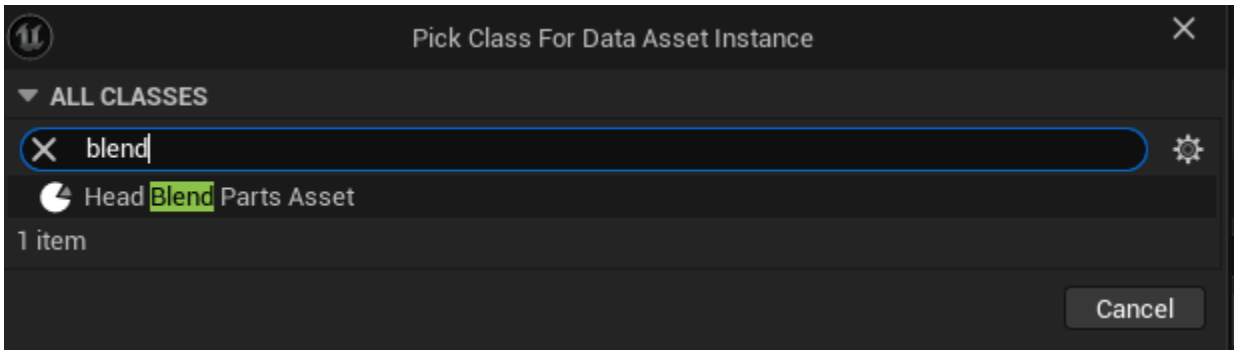
- Подготовить ассет с базовым и целевыми мешами, настроить DNA.
- Можно скачать персонажей из Metahuman



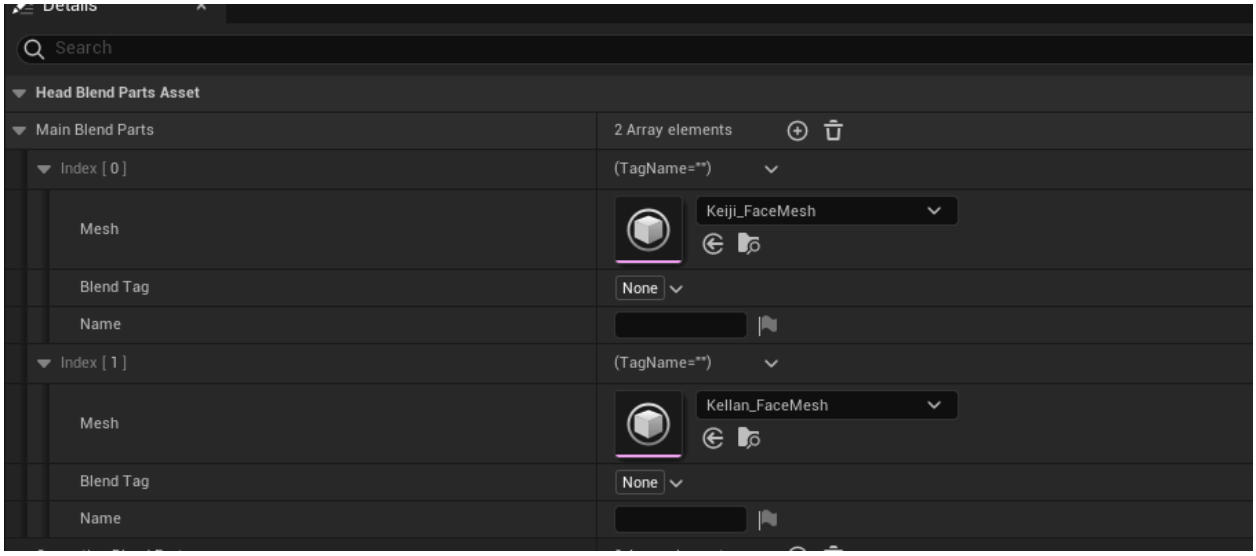
- Или использовать тестовые ассеты, включенные в плагин, из папки DnaClibTool Content/MetaHumans
- Создадим новый блюпринт data asset (head generator asset) и data asset(head blend parts asset)



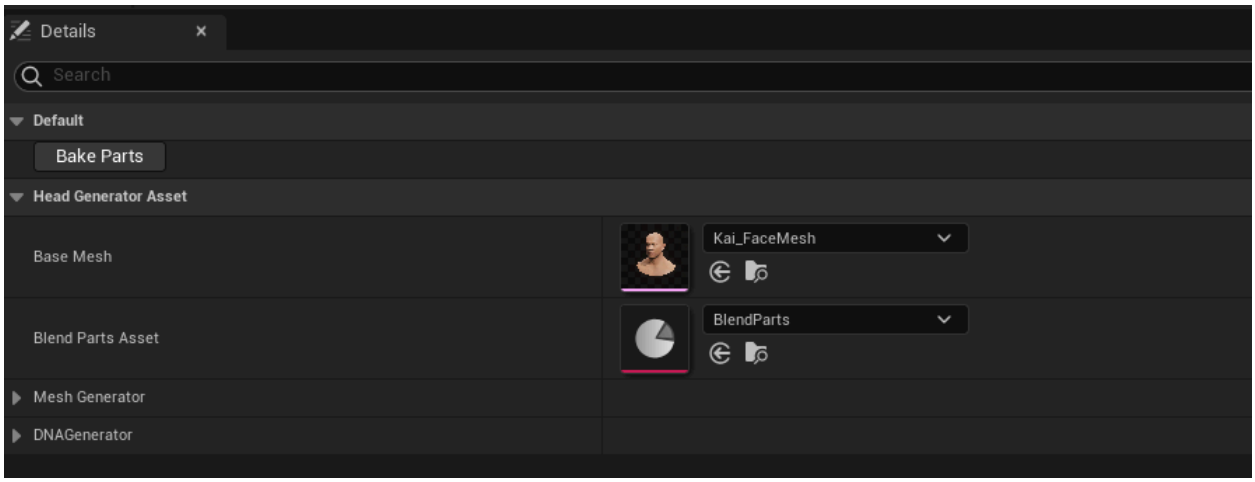




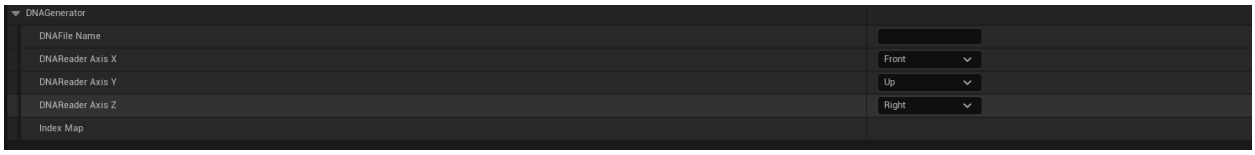
Добавим бленд партсы



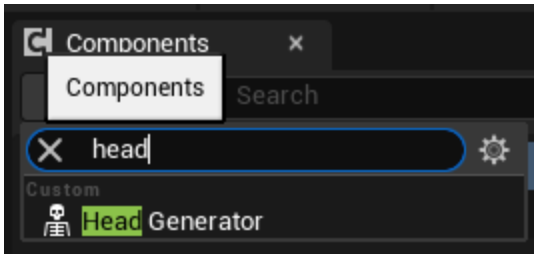
Затем настроим generator asset и нажмем bake parts



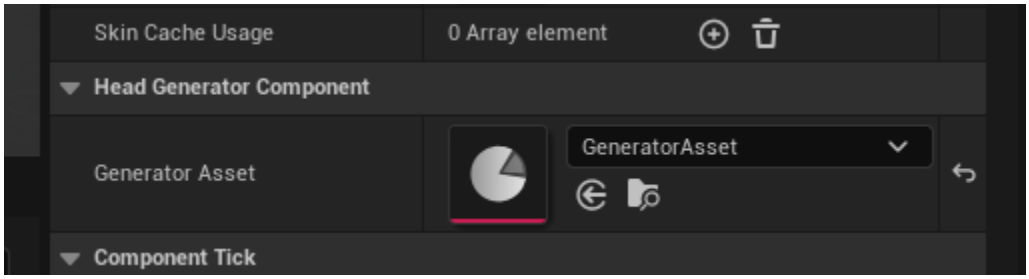
Важно не забыть настроить оси (для метахьюмана как на картинке)



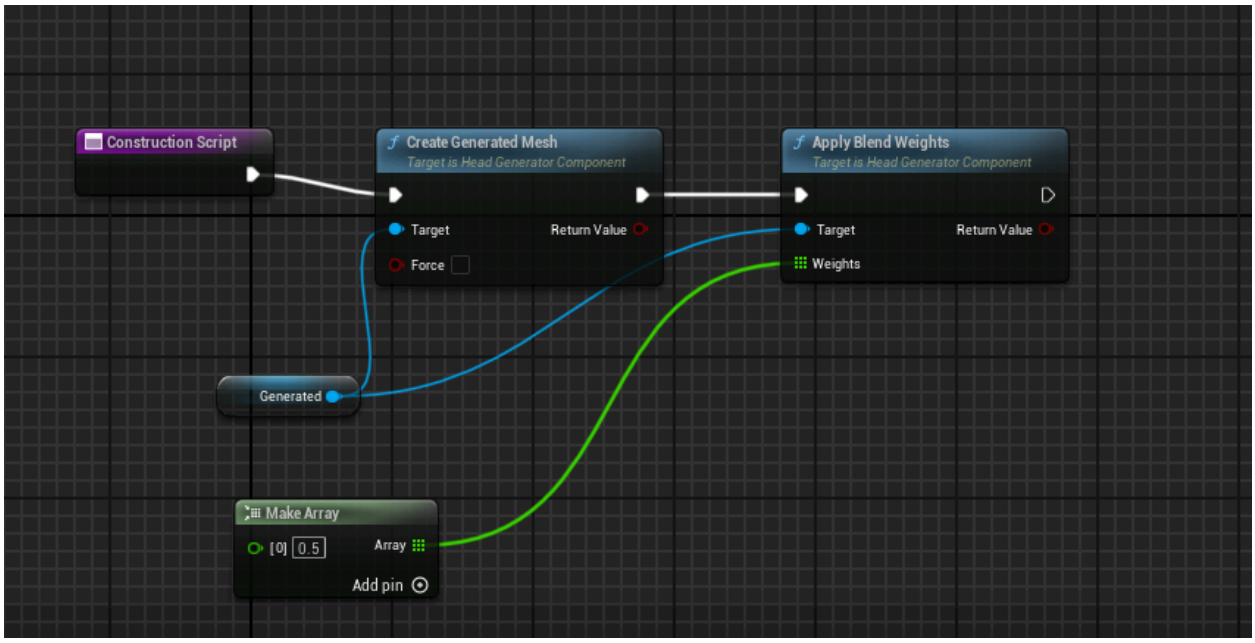
Для актора добавляем компонент (HeadGenerator).



Указываем ему ассет



Использование может выглядеть как-то так



Результате получим что-то среднее между головами (сгенерированый меш в середине)

Базовые плагины

Базовые плагины содержат общий код, использующийся всеми остальными инструментами.

KristaMisc

Плагин KristaMisc содержит чисто служебный код

Как установить плагин в проект

1. Перенести директорию KristaMisc в директорию Plugins проекта.
2. Запустить редактор
3. Перейти в меню Edit -> Plugins
4. Найти плагин KristaMisc и поставить галку напротив
5. Перезапустить редактор

KristaAssertions

Как установить плагин в проект

1. Перенести директорию KristaAssertions в директорию Plugins проекта.
2. Запустить редактор
3. Перейти в меню Edit -> Plugins
4. Найти плагин KristaAssertions и поставить галку напротив
5. Перезапустить редактор

Плагин KristaAssertions содержит систему рантайм-проверок (assertions) позволяющую контролируемо выводить сообщения об ошибках. Плагин имеет настройки, доступные в меню Project Settings/Krista Assertion Settings:

- Show Dialogs On Engine Ensure - превращать вызовы ensure в коде в видимое всплывающее окно
- Show Dialogs On Krista Ensure - выводить всплывающие окна при ошибках в макросах KRISTA_ENSURE
- Never break into debugger - НЕ останавливать отладчик если ошибка случилась при подключенном отладчике
- Assertions Prevented For App Lifetime - ошибки, для которых выбрана опция “не показывать больше”, не будут вызывать всплывающее окно до перезапуска редактора. В противном случае, только до перезапуска игровой сессии
- No Callstacks in Test/Shipping - отключение сбора стеков вызова в конфигурациях Test/Shipping соответственно

Использование

Плагин добавляет макросы (определенные в файлу Ensure.h) KRISTA_ENSURE_* которые вызывают сообщение об ошибке не только в лог, но и (опционально) с показом всплывающего окна и возможностью отменить повторные сообщения о той же ошибке. Эти макросы широко используются в других плагинах Krista Toolset

GameRun

Как установить плагин в проект

1. Перенести директорию GameRun в директорию Plugins проекта.
2. Запустить редактор
3. Перейти в меню Edit -> Plugins
4. Найти плагин GameRun и поставить галку напротив
5. Перезапустить редактор

Плагин добавляет подсистему GameRun для менеджмента времени жизни игровых систем (таких как квесты, диалоги и т.п.) Используется для инициализации других плагинов из Krista Toolset
Для использования необходимо в игровом коде (например в реализации UGameInstance или AGameMode) вызвать

```
GetGameInstance()->GetSubsystem<UGameRunManager>()->StartGameRun();
```

И

```
GetGameInstance()->GetSubsystem<UGameRunManager>()->FinishGameRun();
```

Соответственно.

Collision Detector

Зачем нужен этот плагин, какие задачи решает

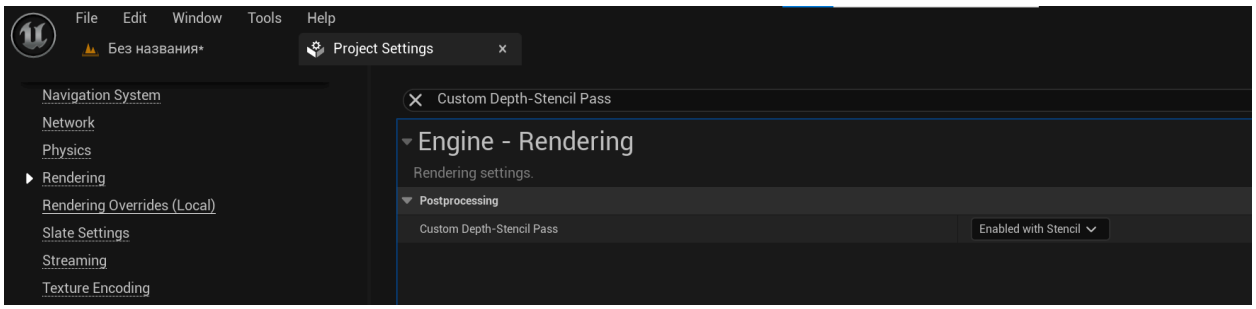
Плагин подсвечивает красным акторы, если они пересекаются. Позволяет расположить объекты на уровне без пересечений, которые могут приводить к “взрывам” во время запуска физической симуляции при запуске PIE и в собранной игре.

Как установить плагин в проект

1. Перенести директорию CollisionDetector в директорию Plugins проекта.
2. Запустить редактор
3. Перейти в меню Edit -> Plugins
4. Найти плагин CollisionDetector и поставить галку напротив
5. Перезапустить редактор

Как настроить плагин для первоначальной работы

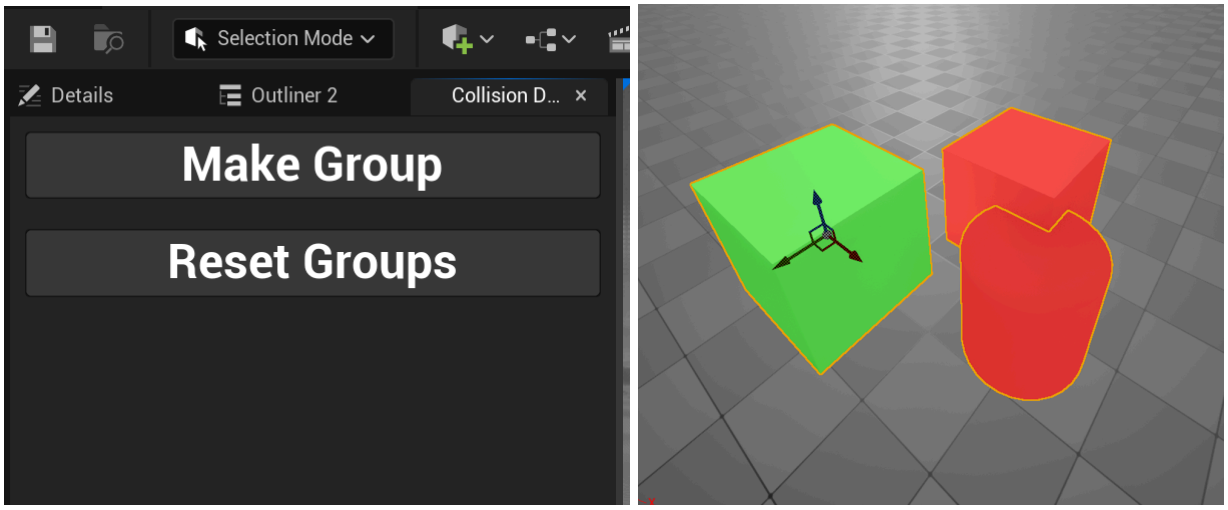
1. Запустить редактор
2. Перейти в меню Edit -> Project Settings
3. Перейти в раздел Engine -> Rendering
4. Установить настройку Custom Depth-Stencil Pass в Enabled with stencil



Окно плагина и интерфейс

Окно плагина открывается в меню Tools -> Collision Detector.
Для удобства появившееся окно плагина можно закрепить в боковой панели.
Интерфейс окна содержит всего 2 кнопки:

Make Group - Выберите несколько акторов во вьюпорте или аутлайнере, а затем нажмите эту кнопку чтобы включить подсветку пересечения для выбранных акторов. Пересекающиеся акторы будут подсвечены красным, непересекающиеся - зелёным.
Reset Groups - Нажмите, чтобы очистить группы, это сбросит всю подсветку.



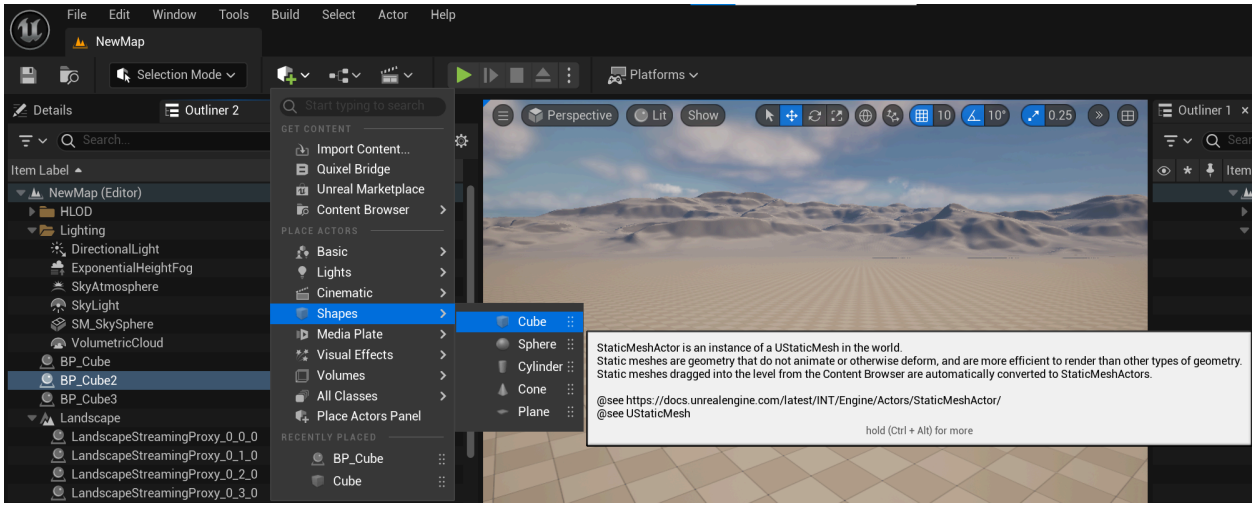
Настройки плагина

Плагин содержит актор CollisionDetector/Actors/BP_CollisionDebuggerActor в котором можно настроить 2 параметра:
Include Subactors - включает подсветку для вложенных акторов
Mark Static Components - Включает подсветку статичных компонент.
При открытии окна плагин создаёт временный актор BP_CollisionDetectorActor, который будет удален при закрытии окна.

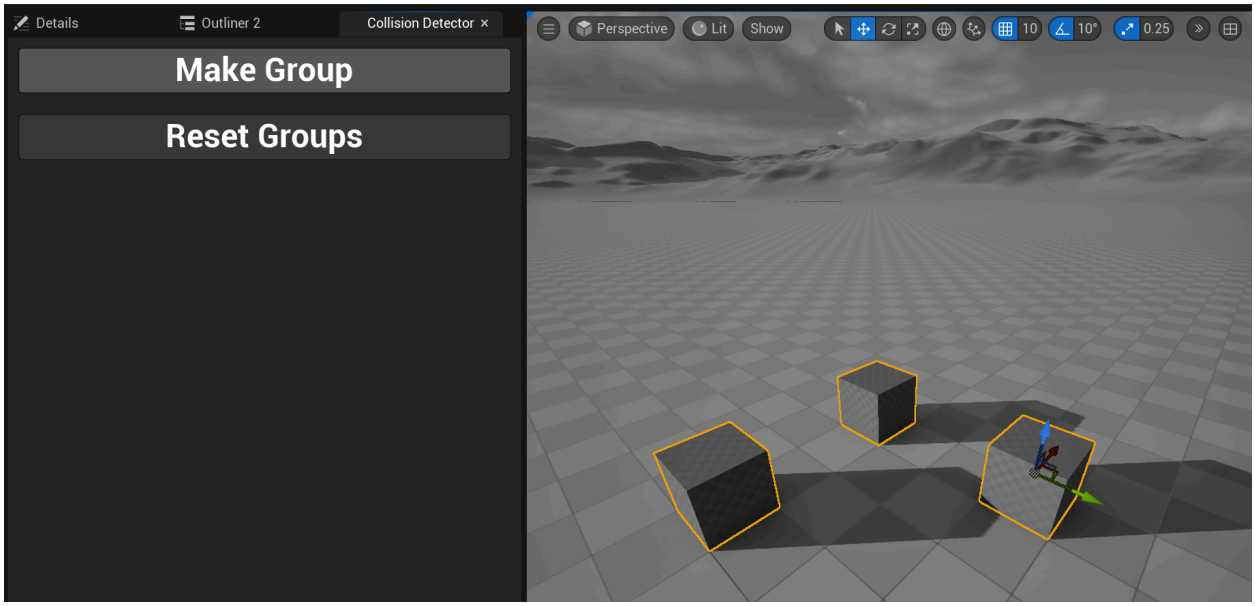
Сценарий использования

Перед использованием обязательно провести настройку в Project Settings (см. выше), иначе подсветка отображаться не будет!

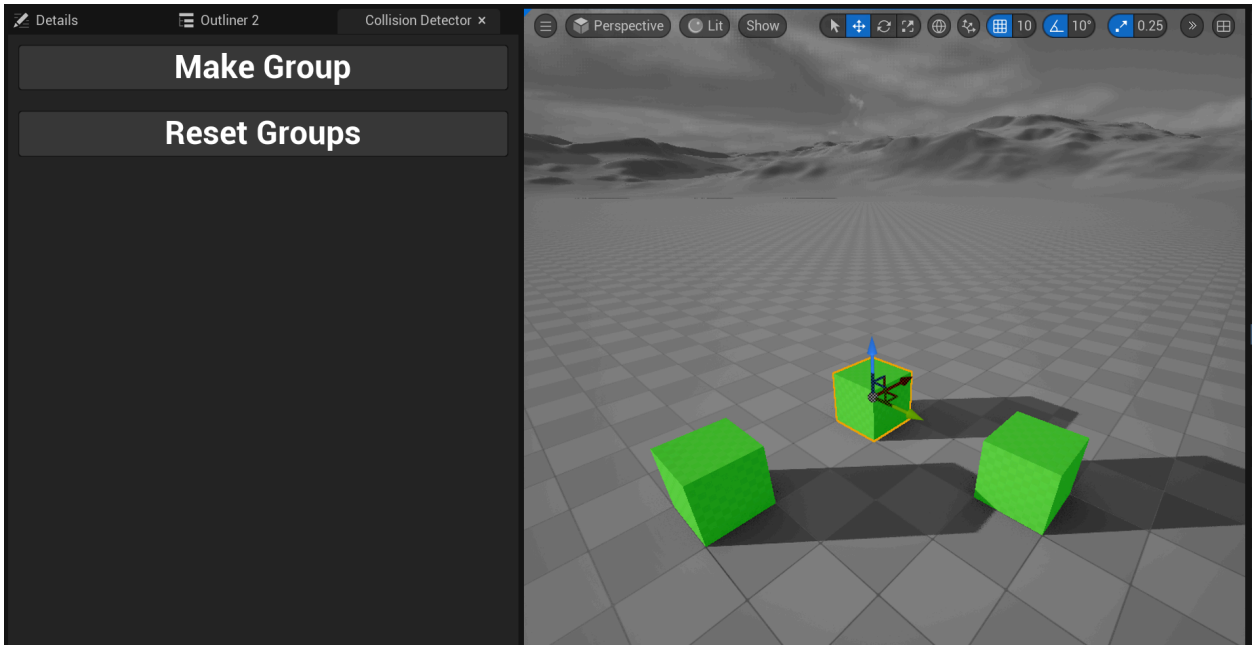
1. Загрузить любой уровень, можно пустой
2. Разместить несколько акторов, имеющих StaticMeshComponent. Например кубы из меню простых форм.



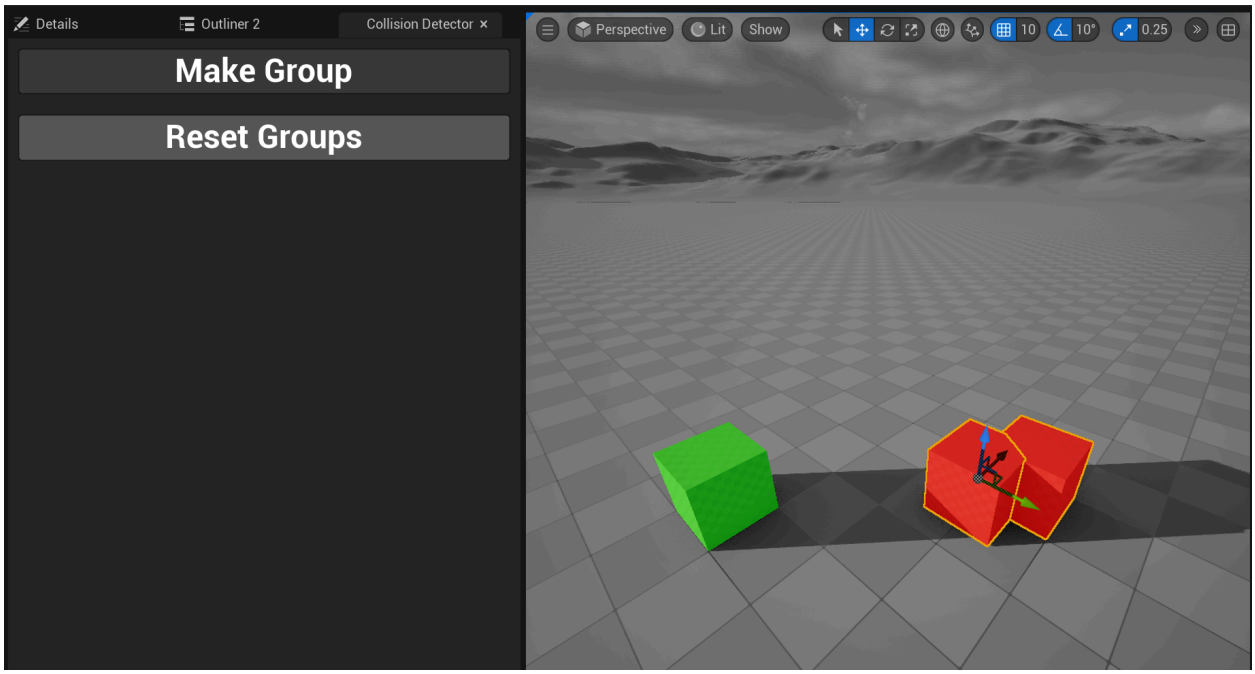
3. Открыть окно плагина через меню Tools -> Collision Detector.
4. Выделить размещённые на карте кубы и нажать Make Group



5. Выделить любой куб, чтобы спровоцировать перерисовку. Все кубы начнут подсвечиваться зелёным, т.к. не пересекаются



6. Сместить выбранный куб так, чтобы он стал пересекаться с другим, оба куба начнут подсвечиваться красным.



Known Issues

Проверялось и работает только со статик мешами.
Точно не работает с акторами, собранными на GeometryCollectionComponent.

Compensator

Зачем нужен этот плагин, какие задачи решает

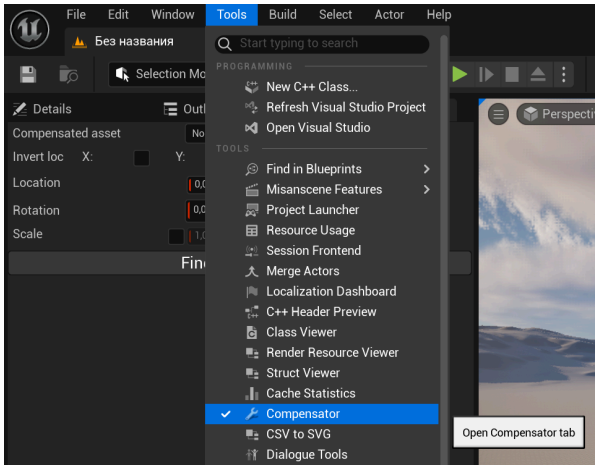
Плагин позволяет вносить массовые изменения вращения, смещения, масштабирования, а также инвертировать оси для экземпляров указанного типа блупринтов во всех картах проекта. Необходим в случаях, когда из-за изменения модели нужно компенсировать её положение в пространстве во всех местах её использования. Плагин работает только на картах со включенным World Partition.

Как установить плагин в проект

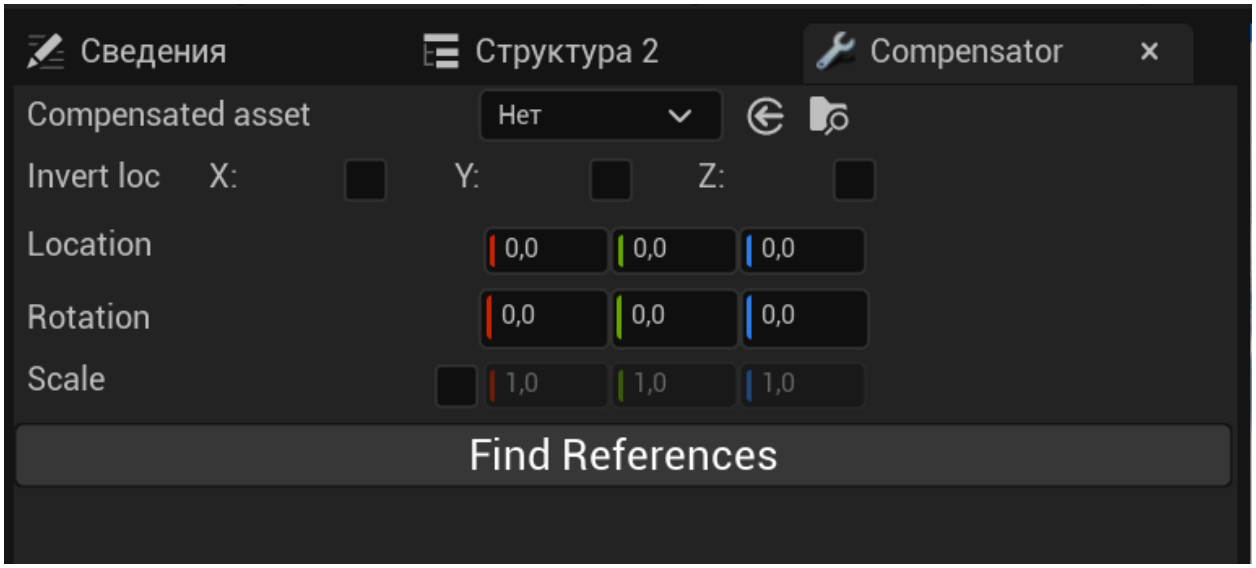
1. Перенести директорию Compensator в директорию Plugins проекта.
2. Запустить редактор
3. Перейти в меню Edit -> Plugins
4. Найти плагин Compensator и поставить галку напротив
5. Перезапустить редактор

Окно плагина и интерфейс

Окно плагина открывается в меню Tools -> Compensator.
Для удобства появившееся окно плагина можно закрепить в боковой панели.

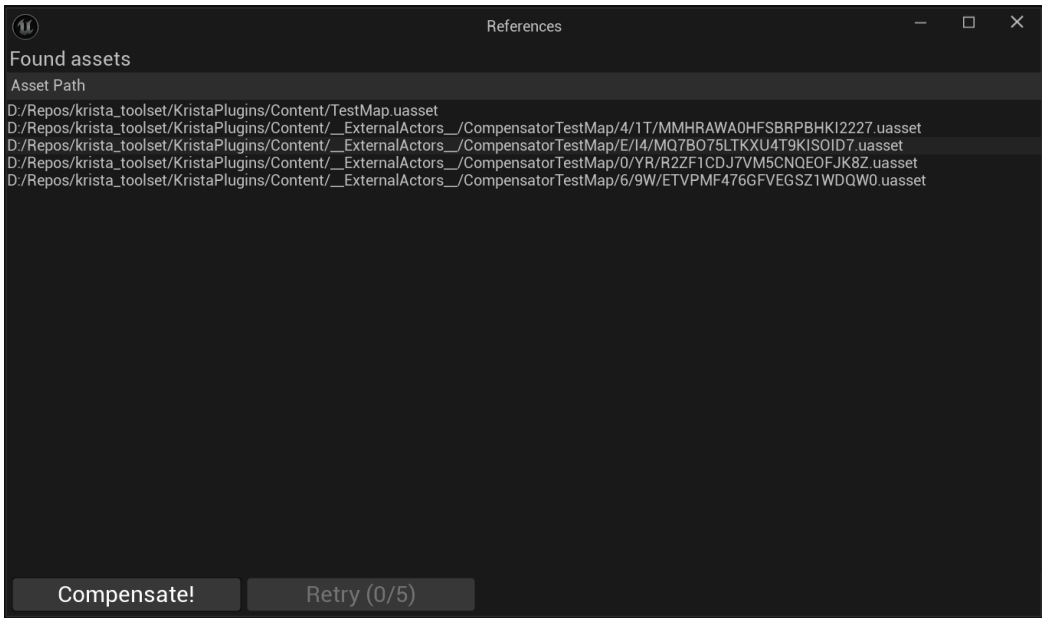


Интерфейс окна содержит следующие элементы:



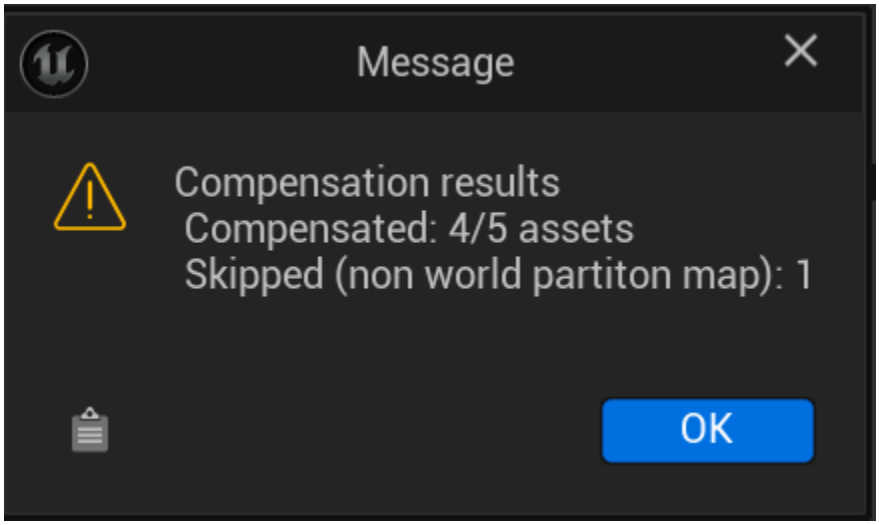
Compensated asset - Поле выбора блуринта, экземпляры которого будут использованы для применения настроек.
Invert loc X Y Z - Настройки инвертирования осей, поставьте галку напротив нужной оси.
Location, Rotation, Scale - Настройки смещения, вращения и масштабирования.

Find References - Данная кнопка запустит поиск экземпляров блупринта и откроет новое окно с результатами поиска - списком найденных экземпляров.



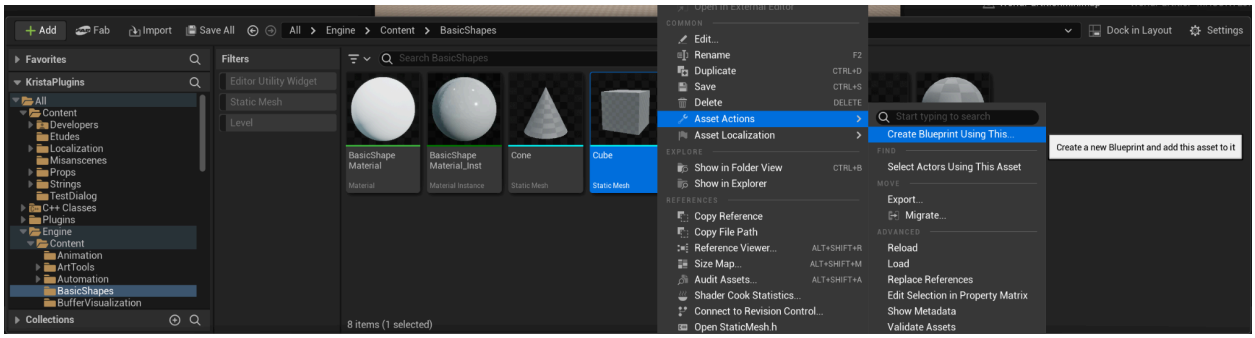
Compensate - В новом окне можно применить настройки ко всем найденным экземплярам нажав кнопку **Compensate.** Состояние процесса будет отображено прогрессбаром с кнопкой cancel.
Retry - Если остановить процесс применения настроек, то его можно будет продолжить с помощью этой кнопки.

Результаты работы после нажатия кнопки Compensate появятся во всплывающем окне

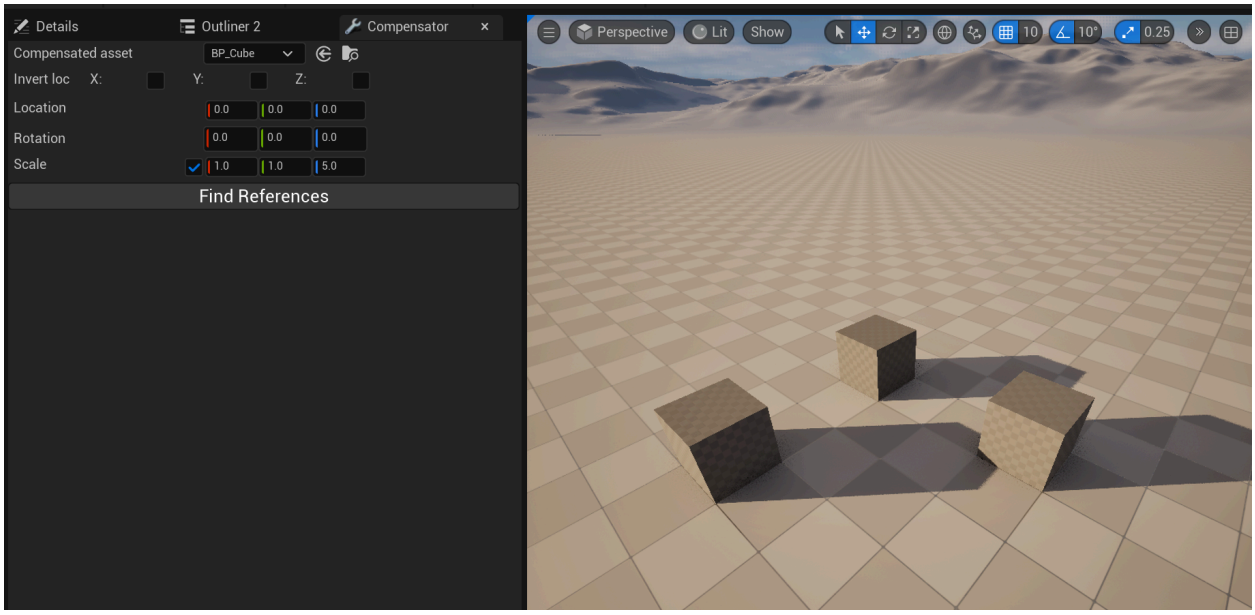


Сценарий использования

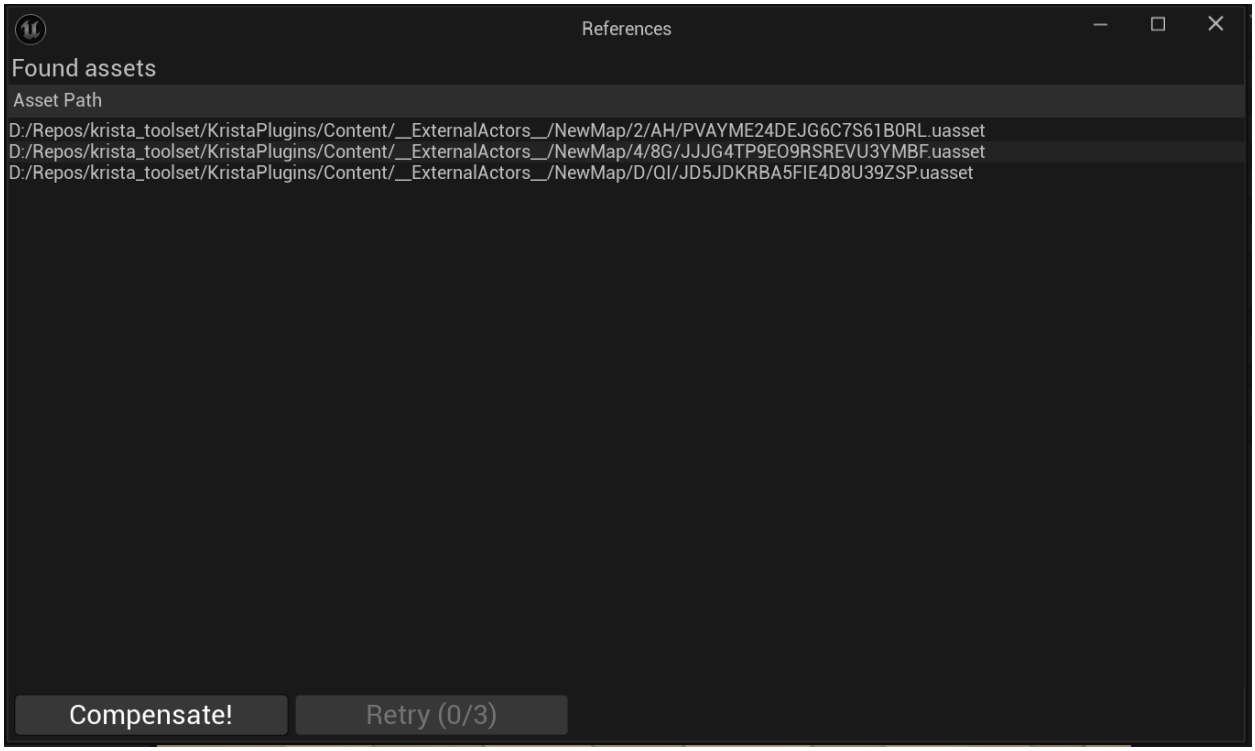
1. Создать блупринт, например выбрать в content browser accet All/Engine/Content/BasicShapes/Cube.uasset и в его контекстном меню выбрать Asset Actions -> Create Blueprint Using This. Сохранить результат, в данном примере назовём блупринт BP_Cube.



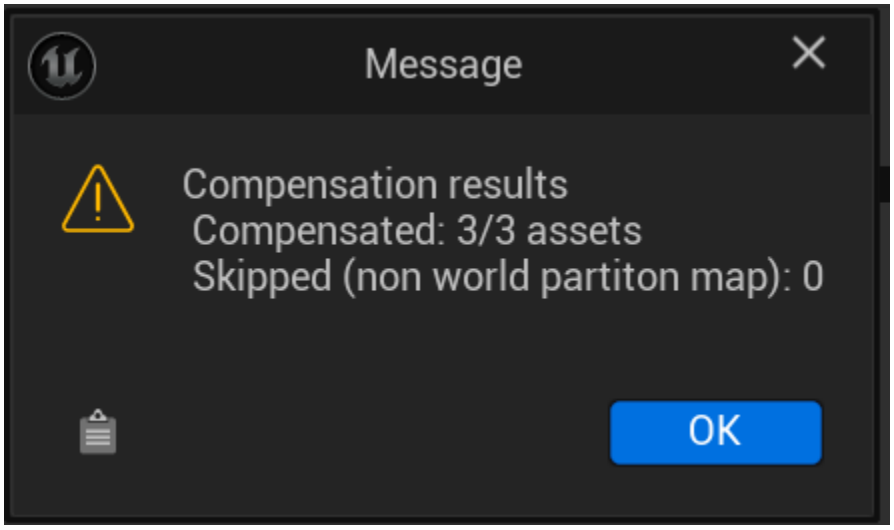
2. Разместить созданный блупринт BP_Cube на уровне, сохранить уровень. Уровень должен иметь включенную настройку World Partition.
3. Открыть окно плагина через меню Tools -> Compensator.
4. Выбрать в поле **Compensated asset** созданный блупринт BP_Cube, (либо другой по необходимости)
5. Задать настройки инвертирования осей, смещения, вращения или масштабирования. В данном примере зададим настройки масштабирования по оси Z, поставив галку в нижнем ряду настроек и указав в соответствующем поле 5



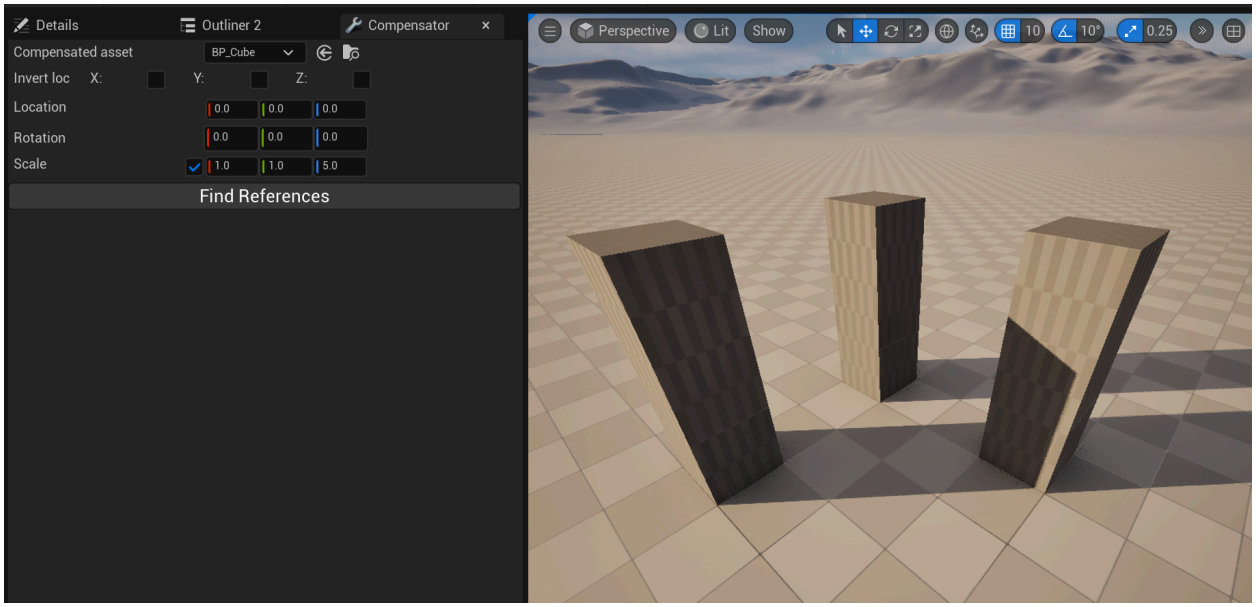
6. Нажать кнопку Find References
7. В появившемся окне нажать кнопку Compensate.



8. Закрыть всплывающее окно с результатами работы,



9. Переключиться во вьюпорт и кликнуть на любой объект на уровне, чтобы спровоцировать перерисовку объектов уровня.
10. Наблюдаем, что все экземпляры указанного блупринта изменили свой масштаб, увеличившись в 5 раз по оси Z.



Зачем нужен плагин?

Плагин предназначен для автоматической замены статических мешей (StaticMesh) на блюпринты (Blueprint). Это нужно для того чтобы контролировать зависимости и централизованно обновлять контент.

У плагина есть следующий функционал:

- Подмена StaticMesh, попадающих на уровень, на соответствующие блюпринты
- Кастомные манипуляции с блюпринтами, которые нужно сделать в момент постановки актора на уровень или в момент его редактирования.
- Консольные команды для репарента блюпринтов

Как работает

При постановке на уровень нового ассета происходит следующее:

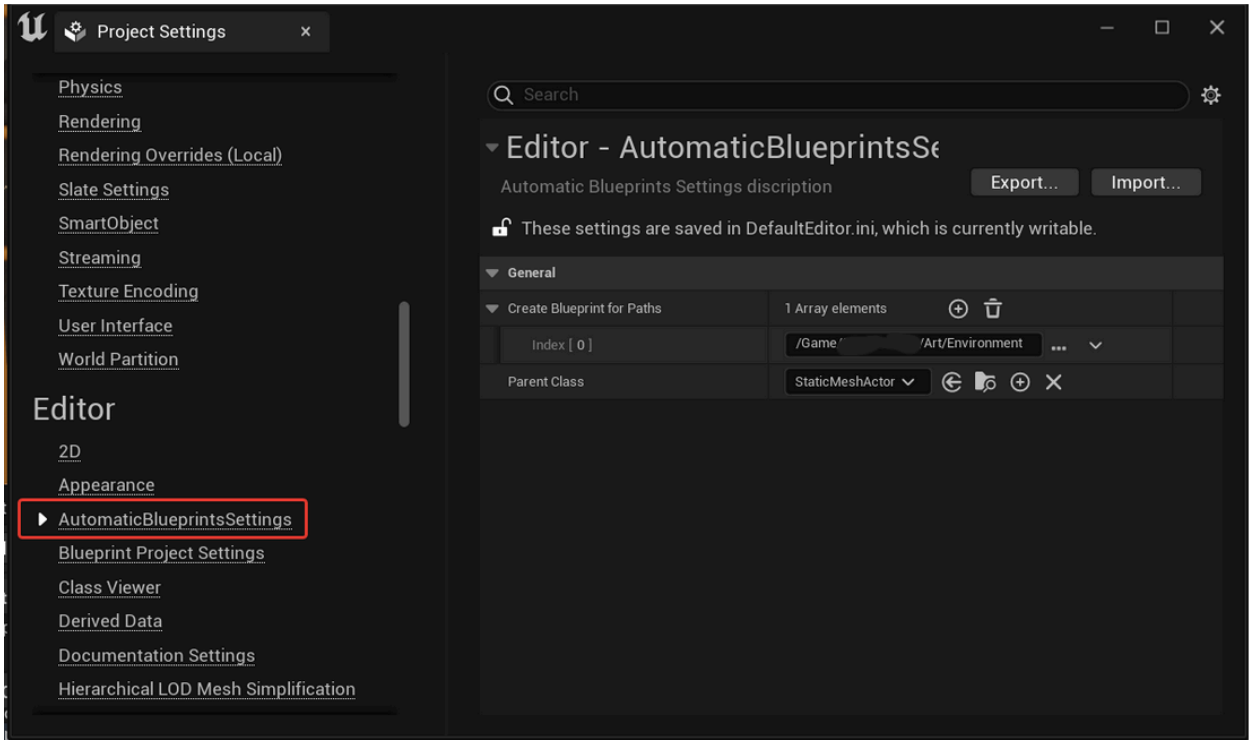
1. Выполняется проверка, что ассет находится внутри одной из директорий, заданных в настройках (CreateBlueprintForPaths)
2. Выполняется проверка - существует ли блюпринт для этого статик меша
    a. Если существует, то берется уже имеющийся блюпринт и ставится на уровень вместо статик меша
    b. Если блюпринта еще не существует, то он создается автоматически (и тоже ставится на уровень).

Как установить плагин в проект

1. Перенести директорию AutomaticBlueprints в директорию Plugins проекта.
2. Запустить редактор
3. Перейти в меню Edit -> Plugins
4. Найти плагин AutomaticBlueprints и поставить галку напротив
5. Перезапустить редактор

Как настроить плагин для первоначальной работы

1. Запустить редактор
2. Перейти в меню Edit -> Project Settings
3. Перейти в раздел Editor -> AutomaticBlueprintsSettings



Необходимо указать следующие параметры в разделе General -> Default Paths Settings

- **CreateBlueprintForPaths** - массив директорий. При постановке статик меша на уровень будет происходить проверка - если ассет находится внутри одной из указанных директорий, то для него будет осуществлена подмена на блюпринт.
- **ParentClass** - тип создаваемых блюпринтов. Сейчас существует техническое ограничение - этот тип должен быть унаследован от StaticMeshActor.

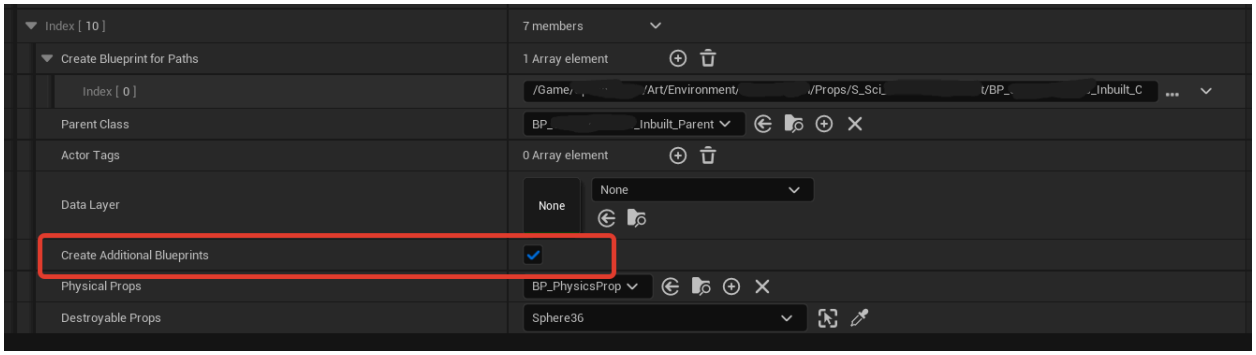
Дополнительно в разделе General -> Paths Settings можно настроить те же параметры для другого набора путей и ParentClass’ов.

Другие настройки

В разделе Edit -> Project Settings -> Editor -> AutomaticBlueprintsSettings также есть следующие настройки:

- **Actor Tags** - тэги которые будут проставляться в созданные блупринты

- **DataLayer** - размещаемые акторы после замены на экземпляр блупринта будут добавляться в указанный даталейер
- **Create Additional Blueprints** - эта галка включает возможность указать дополнительные 2 класса (DestroyablesParentClass и PhysicalParentClass), которые будут использоваться вместо **ParentClass** при размещении акторов на карте, если зажать определённые хоткеи:
    - **DestroyablesParentClass** - будет использован вместо ParentClass при выносе статик меша с зажатой клавишей shift
    - **PhysicalParentClass** - будет использован вместо ParentClass при выносе статик меша с зажатой клавишей Alt



Директории блюпринтов

Сейчас поддержана следующая схема хранения ассетов. Меш лежит в уникальной директории под своим именем и дополнительно упакован в папку Mesh. Блюпринт создаётся в той же директории, но вне папки Mesh, и имеет префикс BP_

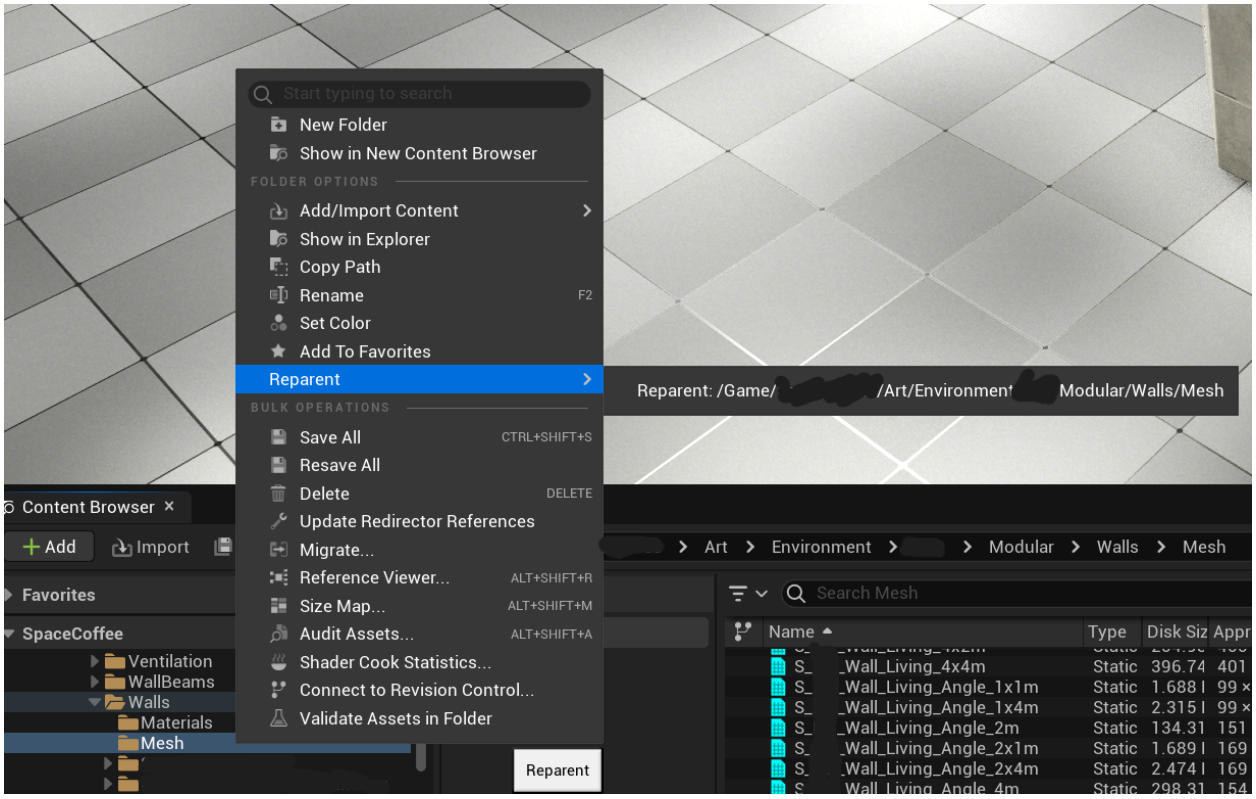
Например:

- Блюпринт: MyProject\Content\MyProject\Art\Environment\Props\S_Ballon**BP_Ballon**
- Статик меш: MyProject\Content\MyProject\Art\Environment\Props\S_Ballon**Mesh****S_Ballon**

Работа модуля опирается на такую структуру файлов при проверках имеющихся блюпринтов и генерации новых.

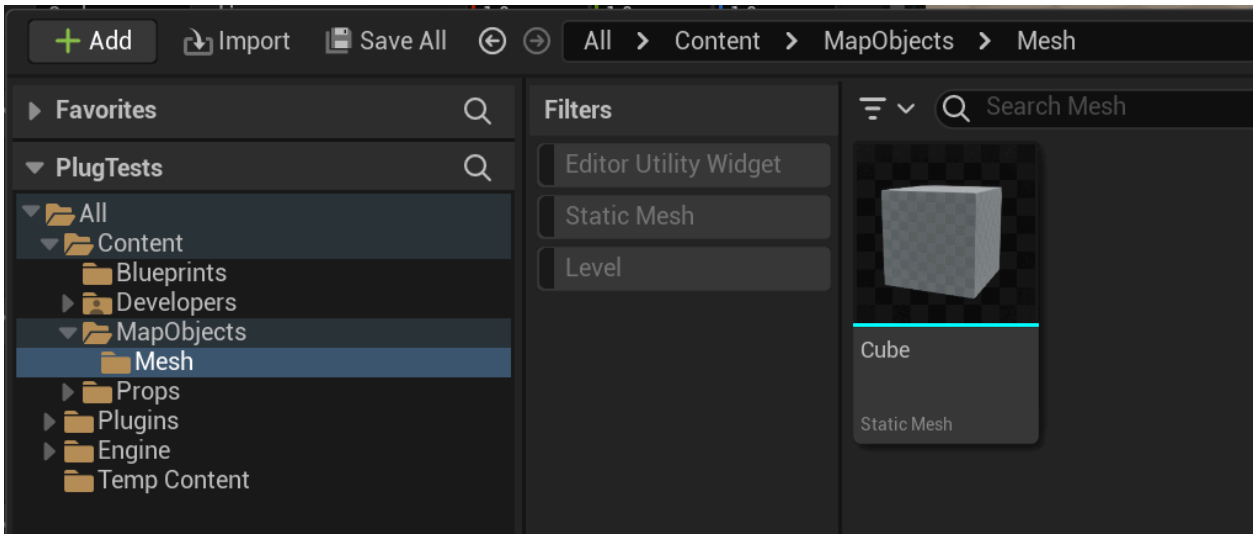
Репарент

Возможен репарент вручную из контекстного меню щелчком по папке.



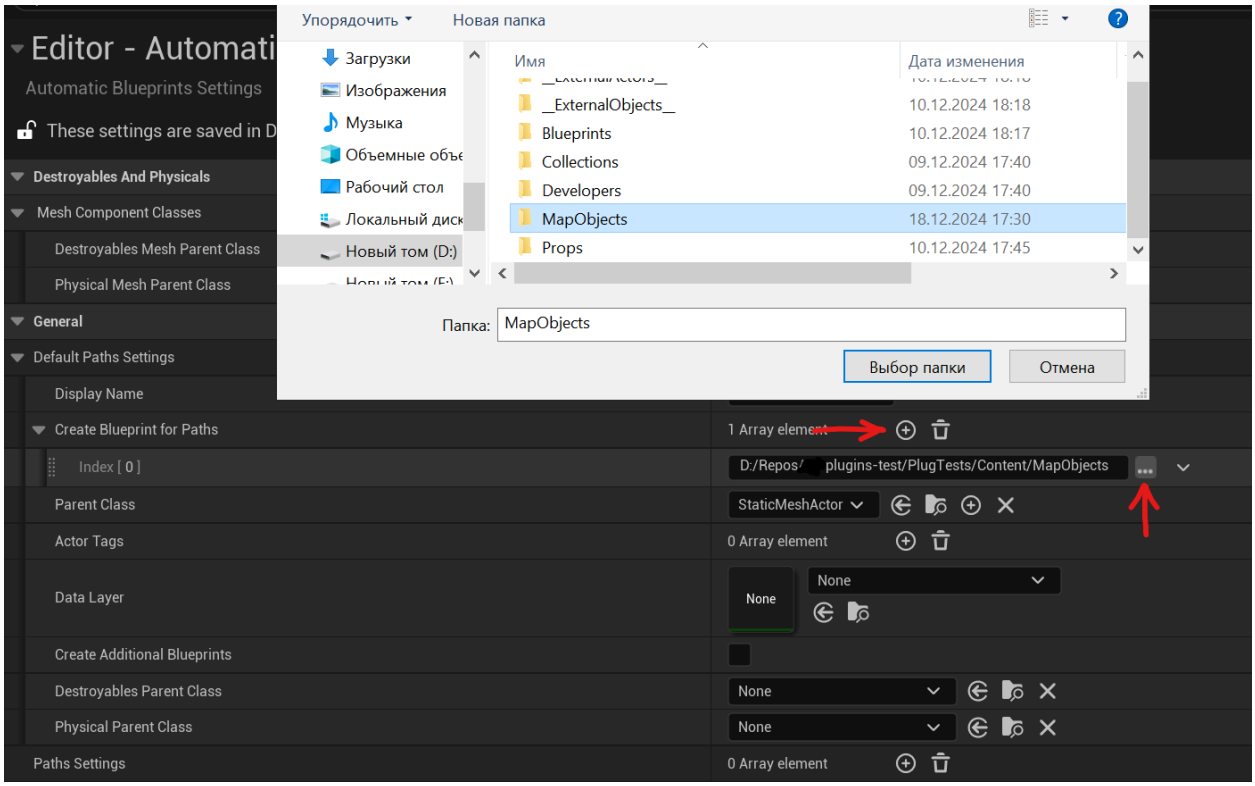
Сценарий использования

1. Установить плагин в проект
2. Создать в проекте директорию Content/MapObjects/Mesh и скопировать туда ассет: All/Engine/Content/BasicShapes/Cube.uasset. Сохранить ассет.

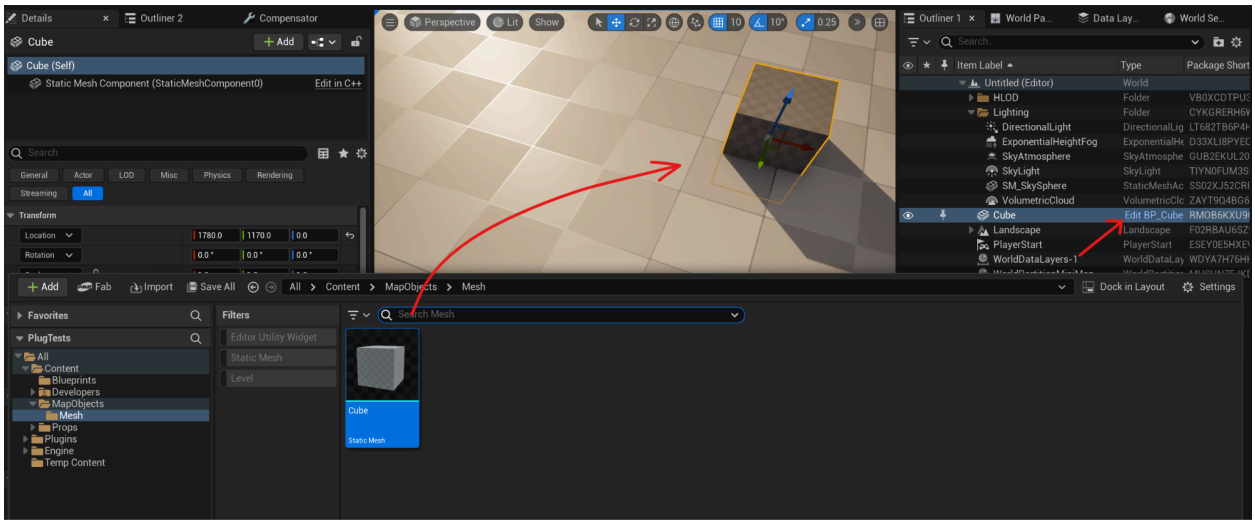


3. Перейти в настройки Edit -> Project Settings -> Editor -> AutomaticBlueprintsSettings

4. В поле CreateBlueprintForPaths нажать на символ “плюс”, в появившемся ниже пустом поле указать путь к директории Content/MapObjects (справа от поля есть символ “...”, можно нажать туда и выбрать директорию во всплывающем окне).
Путь должен быть относительный в формате /Game/…, убедитесь что это так.
Если путь абсолютный (как на скриншоте), то необходимо исправить его: стереть всё до директории Content, а Content заменить на Game.



5. Закрыть настройки, перенести ассет Content/MapObjects/Mesh/Cube.uasset из Content Browser на уровень. Убедиться, что актор на уровне является блупринтом можно в Outliner - в колонке type будет указано Edit BP_Cube. Также в директории MapObjects появится блупринт BP_Cube



Для подготовки версий был разработан **Менеджер Версий**, который позволяет эффективно управлять коммитами, фильтровать их и закрывать версии. Менеджер версий интегрирован с **GitLab** и **Jira**,

Компоненты, с которыми работает менеджер версий:

1. Gitlab - система контроля версий, используемая для управления репозиториями и коммитами.
2. Jira - инструмент для управления проектами и задачами, который используется для привязки коммитов к конкретным задачам.

Краткая инструкция по установке и настройке компонентов описана ниже.

Механизм работы Менеджера Версий:

1. Сбор коммитов:
    - Коммиты могут быть собраны вручную, по **SHA коммита** или по **номеру задачи Jira**.
    - Также существует возможность автоматического сбора коммитов по **Fix version**.
2. Автоматический Cherry - pick
    - Каждый коммит, предварительно отсортированный по дате, автоматически добавляется в нужную ветку с помощью механизма **Cherry-Pick**. Этот процесс позволяет аккуратно переносить изменения из одной ветки в другую, соблюдая последовательность и минимизируя возможные конфликты. Если конфликты всё таки возникли, Менеджер Версий предложит решить их вручную или подтянуть зависимые коммиты
3. Отправка локальных коммитов в удаленный репозиторий
    - Локальные изменения отправляются в удаленный репозиторий, что обеспечивает синхронизацию всех разработок и позволяет команде работать с актуальной версией проекта

Version Manager значительно ускоряет процесс разработки, автоматизируя работу с коммитами и ветками. Теперь, вместо ручной работы с каждым коммитом, можно легко и быстро переносить изменения из одной ветки в другую. Это позволяет сократить время на интеграцию изменений и повысить эффективность работы команды.

Таким образом, Менеджер версий может являться неотъемлемой частью рабочего процесса, обеспечивая гибкость и контроль над версиями, а также улучшая синхронизацию между командами разработки и управления проектами.

User manual

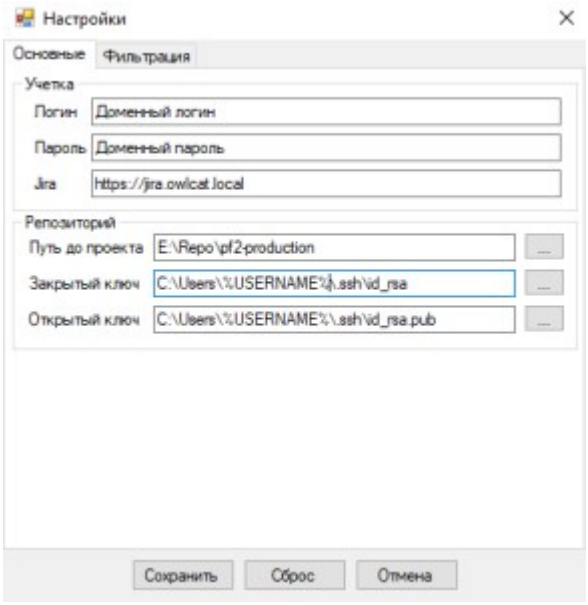
Чтобы его открыть нужно запустить файл ~\VersionManager\Runtime\VersionManager.exe

Настройка

Основные настройки

При первом открытии откроется окно настроек, где нужно настроить в первую очередь основные настройки:

1. Логин - здесь указывается доменный логин.
2. Пароль - здесь указывается доменный пароль.
3. Jira - URL адрес сервера Jira (добавляется по-умолчанию)
4. Путь до проект - полный путь до репозитория с проектом
5. Закрытый ключ - полный путь до закрытого ключа ssh
6. Публичный ключ - полный путь до публичного ключа ssh



После этого можно нажать кнопку *Сохранить*, чтобы создать файл конфигурации.

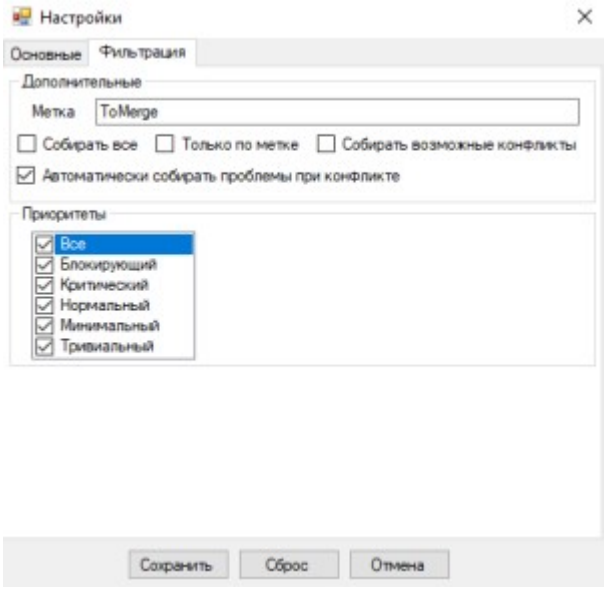
Настройки фильтрации

Дополнительно так же можно настроить фильтрацию задач при сборе коммитов для мержа:

1. Метка - лейбл, по которому будут собираться задача. Работает вместе с включенной галкой *Только по метке*
2. Собирать все - при включении галки будут собираться задачи с любой резолюцией. В противном случае только если резолюция не пуста
3. Только по метке - при включении галки будут собираться только задачи с лейблом указанным в поле *Метка*. Если поставить так же галку *Собирать все*, то задачи с лейблом будут собирать при любой резолюции
4. Собирать возможные конфликты - при включении галки во время поиска коммитов, коммиты, которые прошли проверку фильтрации, будут проверены на возможные конфликты и в случае нахождения таковых, будут собраны в отдельный список для каждого коммита. Без галки коммиты будут собираться быстрее.
5. Автоматически собирать проблемы при конфликте - при включении будут

автоматически собираться конфликтные коммиты при получении конфликта во время мержа. В противном случае при конфликте будет задан вопрос о поиске конфликтных коммитов.

6. Приоритеты - коммиты будут собираться только по задачам, которые соответствуют приоритетам напротив, которых установлены галки. По-умолчанию собираются все приоритеты. Если убрать все галки или установить галку на против *Все*, то будет собираться по-умолчанию.



Процесс мержа коммитов

В качестве подготовки нужно почистить изменения (если они есть), что можно сделать в меню *Репозиторий → Отменить все*. После этого надо перейти на ветку, в которую будут заливаться коммиты из master или другой ветки. Для этого можно использовать меню *Репозиторий → Перейти на ветку.*

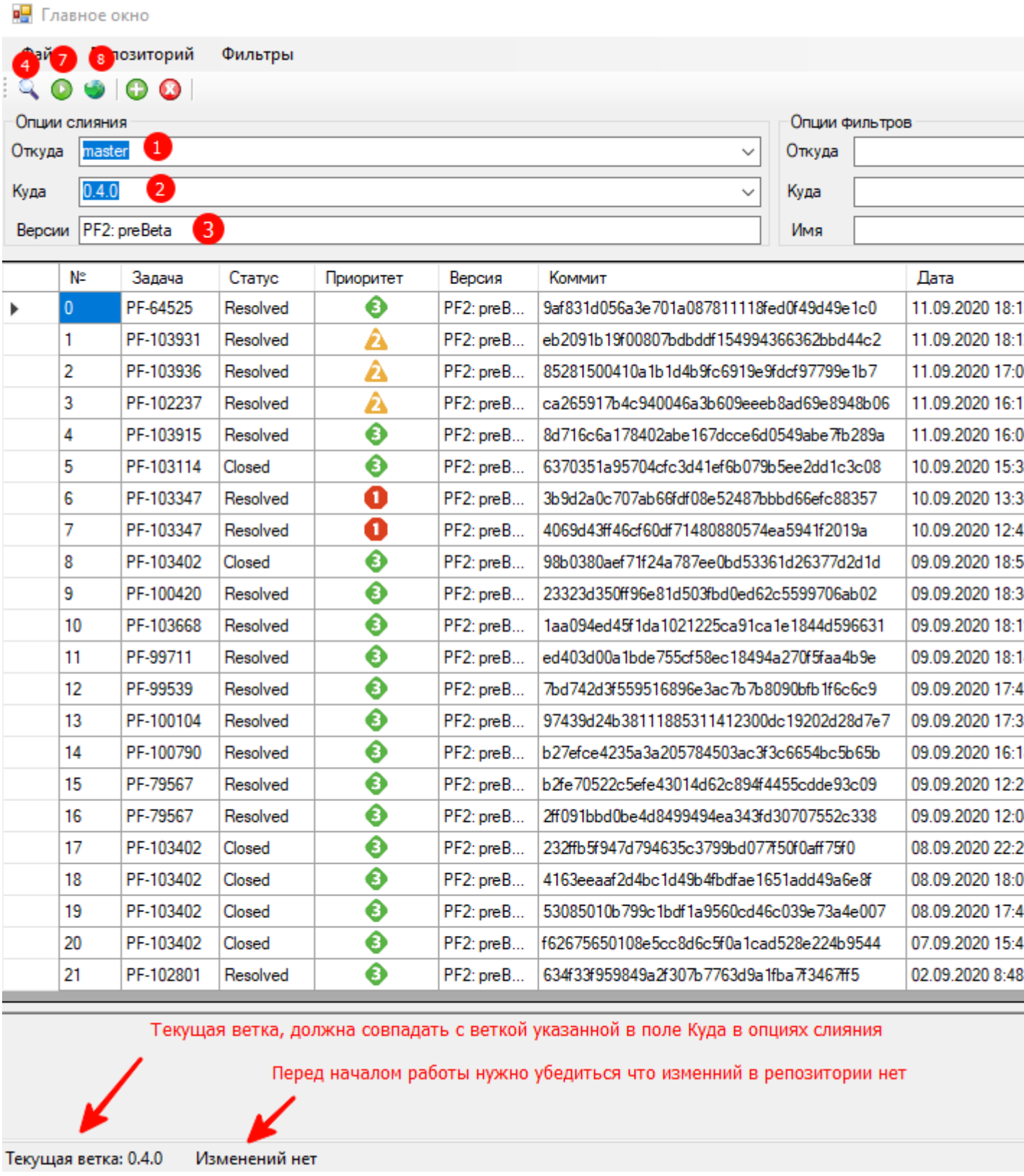
Если репозиторий уже настроен на нужную ветку, то нужно извлечь коммиты из репозитория в меню *Репозиторий → Извлечь*.

Потом следовать алгоритму:

1. Выбрать исходную ветку в списке *Откуда* (для подготовки релизной версии обычно это ветка master)
2. Выбрать конечную ветку в списке *Куда* (ветка, которую был переход при подготовке)
3. Указать fixVersions задач для фильтрации коммитов, которые будут залиты в конечную ветку в поле *Версии* (для подготовки релизной версии обычно используются имена типа Hotfix 1.1.1, PF2: preBeta и т.д.). Если нужно собрать с нескольких версий, то их можно указать через запятую.
4. Нажать на кнопку сбора коммитов (пункт 4 на рис. низу). Скорость процесса зависит от разницы в количестве коммитов между исходной и конечными ветками, а так же от настроек фильтрации.
5. По окончанию процесса сбора коммитов для мержа заполниться таблица. Если была установлена галка *Собирать возможные конфликты* в настройках, то можно увидеть в каких коммитах могут быть конфликты (пункт 5 на рис. выше). Так же можно добавить конфликтные коммиты в таблицу *ПКМ → Открыть конфликты → Выбрать*

конфликтный файл → Выбрать конфликтный коммит → ПКМ по конфликтному коммиту → Добавить.

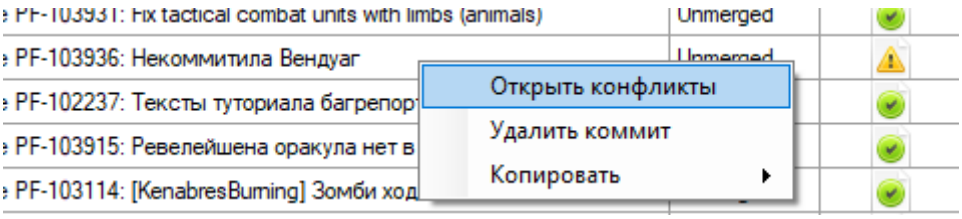
6. Если какие-то коммиты надо исключить из заливки, то можно поставить галочку в колонке *Пропустить* в строке этого коммита (пункт 6 на рис. выше)
7. После проверки списка заливаемых коммитов, нажать на кнопку заливки (пункт 7 на рис. выше). Скорость процесса зависит от количества и размера заливаемых коммитов, например, 10 коммитов с правками по 1 символу зальются примерно так же быстро как 1 коммит с 10 картинками.
8. По окончанию процесса заливки нужно отправить коммиты в удаленный репозиторий, нажав кнопку пуша коммитов (пункт 8 на рис. выше). После пуша всем задачам по замерженным в таблице коммита будет установлен лейбл *Merged*



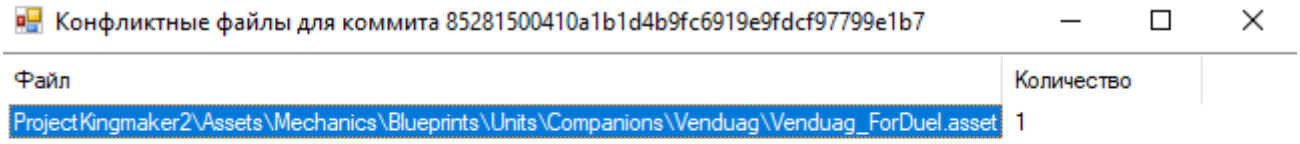
Добавление конфликтных коммитов перед мержом

Перед началом мержа можно добавить коммиты, которые возможно вызовут конфликт. Для это надо:

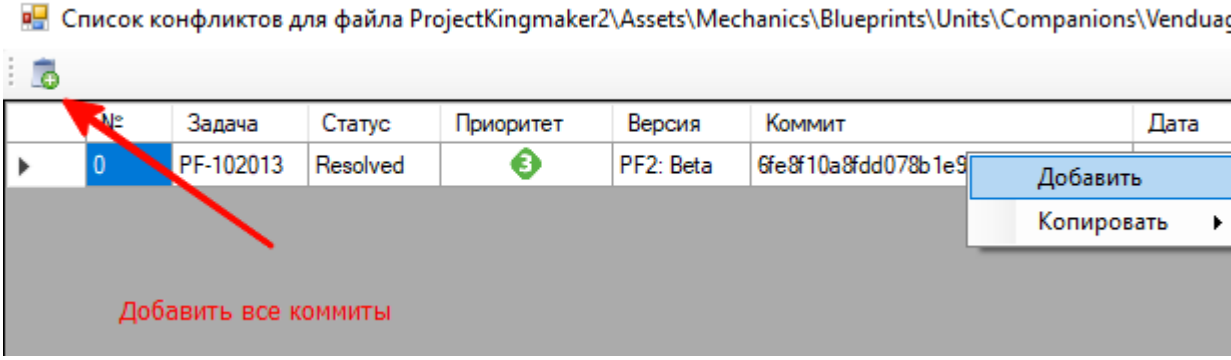
1. Нажать ПКМ на строку, со восклицательным знаком в колонке *Конфликты* и выбрать пункт *Открыть конфликты*



2. В открывшемся окне дважды щелкнуть по пути конфликтного файла, чтобы открыть список конфликтных коммитов



3. В списке конфликтных коммитов для конфликтного файла есть 2 вариант добавления:
    1. Нажать *ПКП* на строке выбранного коммита и выбрать пункт *Добавить*
    2. Нажать кнопку *Добавить все* на панеле инструментов



Решение конфликтов

Если при мерже случился конфликт, то приложение предложит собрать конфликтные коммиты по конфликтным файлам, либо соберет их автоматически, если установленная галка *Автоматически собирать проблемы при конфликте.* После чего процесс мержа остановится и можно будет добавить эти коммиты в таблицу для мержа. Если какие-то коммиты или файлы в них нельзя заливать, то есть несколько вариантов:

1. Так как после конфликта файлы коммита будут висеть модифицированными, то убрать из индекса те файлы которые нельзя заливать и залить остальные. Это можно делать только при условии, что точно известно, что данная операция ничего не сломает. Требует решения автора коммита
2. Можно залить коммит с изменениями из исходной ветки, пропустив изменения из конфликтных файлов. Такой вариант может быть в случае если автор коммита согласен и уверен, что это ничего не сломает.
3. Можно пропустить коммит и не заливать его. Такой вариант возможен, если конфликтные изменения являются устаревшими, например, при заливке локализации. когда последние правки уже были залиты по закрытому таску, а старые стали доступны только после закрытия другого таска, в следствие чего в ветке уже есть все эти изменения.
4. Так же пропустить, если автор коммита решит, что так необязателен при подготовке версии и его можно пропустить.

Работа с fixVersion/s

При релизе версии её необходимо перевести в статус Released. В Archived до конца подготовки ветки переводить не надо, так как Version Manager показывает только неархивные версии, а разработчики вполне могут подлить коммит по какой-то задаче с более ранней версией и этот коммит может потеряться.

Развертывание, подготовка и настройка тестового окружения

Docker-образ Jira

Полная официальная документация по установке Jira в docker находится здесь https://community.atlassian.com/t5/Jira-articles/How-to-run-Jira-in-a-docker-container/ba-p/752697

Ниже описаны основные шаги

1. Подготовка окружения

1) Убедитесь, что на вашем компьютере установлен Docker. Если нет, установите Docker.

2)Убедитесь, что у вас достаточно ресурсов (рекомендуется минимум 4 ГБ оперативной памяти).

2. Создание Docker-сети

Чтобы Jira могла взаимодействовать с другими контейнерами (например, базой данных), создайте пользовательскую сеть Docker:

```
docker network create jira-
network
```

3. Запуск базы данных (PostgreSQL)

Jira требует базу данных. Запустим PostgreSQL:

```
docker run -d \

  --name jira-db \

  --network jira-network \

  -e POSTGRES_USER=jirauser \
```

```
  -e POSTGRES_PASSWORD=securepassword \

  -e POSTGRES_DB=jiradb \

  -v
jira-db-data:/var/lib/postgresql/data \

  postgres:13
```

4. Запуск Jira

Запустите Jira с привязкой к базе данных:

```
docker run -d \

  --name jira \

  --network jira-network \

  -e ATL_DB_TYPE=postgresql \

  -e ATL_DB_HOST=jira-db \

  -e ATL_DB_PORT=5432 \

  -e ATL_DB_USER=jirauser \

  -e
ATL_DB_PASSWORD=securepassword \

  -e ATL_DB_NAME=jiradb \

  -p 8080:8080 \

  atlassian/jira-software:latest
```

5. Настройка Jira

1) Откройте браузер и перейдите по адресу: http://localhost:8080.

2) Следуйте инструкциям мастера установки:

- Укажите тип установки (обычно "Custom").
- Подключите базу данных PostgreSQL.
- Настройте учетные данные администратора.

Docker-образ GitLab

Полная официальная документация по установке Gitlab в docker находится здесь https://docs.gitlab.com/ee/install/docker/installation.html

Ниже написана краткая инструкция

1. Подготовка окружения

1) Убедитесь, что Docker установлен.

2) GitLab требует больше ресурсов. Рекомендуется выделить минимум 4 CPU и 4 ГБ оперативной памяти.

2. Создание Docker-сети

Создайте отдельную сеть для GitLab:

```
docker network create gitlab-
network
```

3. Запуск GitLab

Запустите GitLab с помощью следующей команды:

```
docker run -d \

  --name gitlab \

  --network gitlab-network \

  -p 443:443 \

  -p 80:80 \

  -p 22:22 \

  -e GITLAB_OMNIBUS_CONFIG="external_url
'http://localhost';" \

  -v gitlab-config:/etc/gitlab \

  -v gitlab-logs:/var/log/gitlab \

  -v gitlab-data:/var/opt/gitlab \

  gitlab/gitlab-ce:latest
```

4. Настройка GitLab

1. Откройте браузер и перейдите по адресу: `http://localhost`.
2. На первой странице введите пароль для администратора.
3. Войдите под учетной записью `root` с заданным паролем.
4. Настройте группы, проекты и пользователей при необходимости.

Примечания

1. Убедитесь, что порты 8080 (для Jira) и 80 (для GitLab) свободны перед запуском.
2. Если вы работаете на сервере с ограниченными ресурсами, уменьшите лимиты памяти и процессора с помощью флагов `--memory` и `--cpus`.

Создание проекта, добавление задач и фиксверсии (Fix Version) в интерфейсе Jira

Создание проекта:

1. Зайдите в интерфейс Jira под учетной записью с правами администратора.
2. На главной странице Jira нажмите на кнопку **"Projects"** в верхнем меню.
3. В выпадающем списке выберите **"Create project"**.
4. Jira предложит несколько шаблонов проектов. Выберите Company-managed, так как только они поддерживают настройку фиксверсий по умолчанию
5. Настройте проект, заполнив поля
    - Project name
    - Key: PROJ
    - Project type
    - Access
6. Нажмите кнопку **"Create"**

Добавление задач:

1. Перейдите в созданный проект.
2. На главной странице проекта нажмите кнопку **"Create"** в верхнем меню или в правом верхнем углу.
3. В открывшейся форме укажите:
    - **Issue Type**: Например, Story, Bug или Task.
    - **Summary**: Краткое описание задачи.
    - **Description**: Подробное описание задачи.
    - **Assignee**: Назначьте задачу ответственному сотруднику.

- **Priority**: Укажите приоритет задачи (например, High, Medium, Low).

4. Нажмите **"Create"**, чтобы сохранить задачу.
5. Для тестирования Version Manager, переведите статус задачи в Resolved или Closed

Добавление фиксверсии:

1. Зайдите в интерфейс Jira. На главной странице выберите проект, в котором необходимо добавить фиксверсии.
2. В левом меню проекта выберите пункт **"Releases"**
3. На странице управления версиями нажмите кнопку **"Create version**"
4. Заполните поля:
    - Version name
    - Start date
    - Release date
    - Description
5. Привяжите задачу к фиксверсии:
    - Откройте задачу и в правой панели задач найдите поле "Fix Version/s"
    - Нажмите на это поле и выберите из выпадающего списка необходимую версию
    - Если версия отсутствует в списке, убедитесь, что она была создана
    - Нажмите "Save".

Примечания

1. Поле **"Fix Version/s"** доступно только для задач типов, где это предусмотрено схемой полей и настраивается пользователем администратора.
2. Для массового добавления задач в фиксверсию используйте функцию **Bulk Edit**:
    - Отметьте задачи в списке.
    - Выберите действие **Edit issues**.
    - В открывшемся окне измените значение поля **Fix Version/s**.

Создание репозитория, добавление веток и коммитов

Репозиторий

1. Перейдите в веб интерфейс gitlab
2. На главной странице нажмите New project → Create blank project
3. Задайте имя репозитория и настройте остальные параметры

Ветки

1. Перейдите в созданный репозиторий.
2. В верхней части страницы, рядом с текущей веткой, нажмите на выпадающее меню и

выберите **"New branch"**.

3. Введите имя новой ветки и выберите, от какой ветки будет происходить ее создание (например, от **master** или **main**).
4. Нажмите **"Create branch"**.

Коммиты

Для работоспособности Version manager, в репозитории gitlab сообщения к коммитам должны содержать номер задачи в формате "feature [проект]-[номер_задачи]".
Пример:

```
feature PROJ-1234: New cool
feature
```

Создайте несколько коммитов по задачам Jira в одну из веток прямо в gitlab, следуя инструкции:

1. Перейдите в репозиторий.
2. Нажмите **Edit → Web IDE.**
3. Внесите изменения в репозиторий и сделайте коммит, следуя необходимым стандартам оформления сообщений.

Пользовательская история

Задача

```
Необходимо собрать коммиты по фикс-версии Hotfix 1.1.1 из ветки master и залить их в рел
1.1.1.
```

Решение

1. Подготовка

1. Убедитесь, что репозиторий и профиль Jira настроены корректно:
    - Логин и пароль указаны.
    - Путь до проекта прописан.
    - Установлены пути до закрытого и публичного SSH-ключей.
2. Обновите локальный репозиторий переключитесь на нужную ветку:
    - Репозиторий → извлечь
    - Репозиторий → перейти на ветку → *master*

2. Настройка веток и фильтрации

1. В главном меню выберите ветки откуда мержить (*master*) и куда (*release/Hotfix-1.1.1*.).
2. Установите фильтрацию по фикс-версии:

- В поле *версии* укажите *Hotfix 1.1.1*

3. Сбор коммитов

1. Нажмите кнопку Поиск коммитов и дождитесь завершения процесса.
2. Проверьте список собранных коммитов:
    - Убедитесь, что все нужные задачи отображаются в таблице.

4. Заливка коммитов

1. Нажмите кнопку "*Слияние коммитов*".
2. Дождитесь завершения процесса. Проверяйте логи для выявления возможных ошибок.

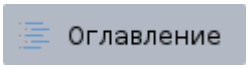
5. Решение конфликтов (если есть)

1. Щелкните ПКМ по строке с конфликтом и выберите *Открыть конфликты*.
2. Добавьте конфликтные коммиты в таблицу:
    - По отдельности или нажмите *Добавить все*.
3. Убедитесь, что конфликтные файлы обработаны:
    - Отключите индексацию ненужных изменений.
    - Пропустите ненужные коммиты, если это безопасно.
4. Повторите шаг 4 "Заливка коммитов"

6. Публикация изменений

1. Нажмите кнопку Отправка коммитов, чтобы отправить изменения в удаленный репозиторий.
2. Убедитесь, что все задачи, связанные с залитыми коммитами, помечены как "Merged".

Теперь релизная ветка *release/Hotfix-1.1.1* содержит все нужные коммиты из фикс-версии *Hotfix 1.1.1*. 🎉



Krista Bugreport

Краткое описание:

Krista Bugreport - инструмент, для удобной отправки на сервер информации о текущем состоянии игры и автоматического создания тикетов в Jira. Багрепорт очень удобен во время разработки, поскольку позволяет сотрудникам (QA и не только) сообщить о имеющейся проблеме, не затрачивая на это лишнего времени и без отрыва от основной деятельности. Этот инструмент также может быть полезен для вышедшего продукта, если вы хотите дать пользователям инструмент для легкого сообщения о возникающих ошибках, чтобы они могли быть оперативно решены.

Данное решение, легко интегрируется в любое приложение на Unreal Engine. При поддержке *Siren* со стороны бэкенда *Криста Багрепорт* может быть использован для автоматического генерации тикетов в Jira сразу со всей необходимой информацией в теле, а также автоматическим заполнением полей.

Три основные части реализации баг репортера:

- Модуль баг репортера **KristaBugreport** и подсистема **BugreportSubsystem**
    - Вынесенная в отдельный модуль подсистема для доступа к объектам отвечающим за создание и отправку репортов об ошибках
- Реестр игровых контекстов **ContextRegistry**
    - Этот механизм предназначен для передачи произвольных данных об игровых системах, которые есть необходимость закинуть в репорт. Реестр организован таким образом, чтобы разные игровые системы могли добавлять информацию о себе в отчет независимо друг от друга и от плагина багрепорта.
- Механизм взаимодействия с подсистемой багрепорта со стороны игры.
    - Конкретный интерфейс взаимодействия с игроком/разработчиком гибко настраивается на стороне игры

Добавление плагина в проект

Для добавления плагина в проект необходимо

- Расположить папку плагина **KristaBugreport** в папке Plugins
    - YourProject/Plugins/

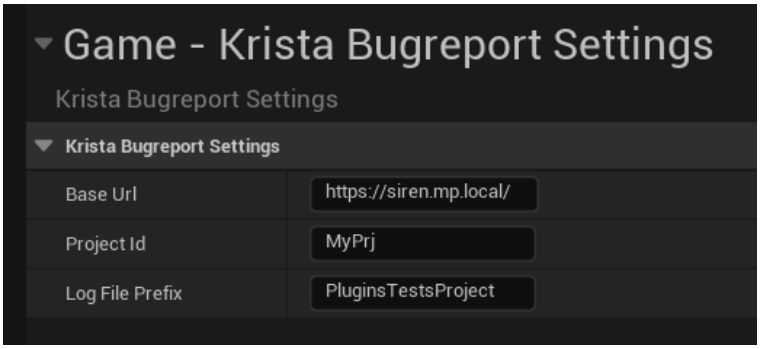
    - Туда же поместите плагин **RuntimeArchiver**, который необходим для работы **KristaBugreport**.
- Прописать плагин в **.uporject** файла вашего проекта

```
"Plugins": [
    {
        "Name": "KristaBugreport",
        "Enabled": true
    },
    {
```

- Добавить зависимость с в **.Build.cs** файл того модуля, из которого будет использоваться плагин

```
PublicDependencyModuleNames.AddRange(new string[] {
        "KristaBugreport",
        "Core",
        "CoreUObject",
```

- Перезапустите IDE.
    - Замечание: В случае открытия проекта через **.sln** файл решения не забудьте перегенерировать файл из **.uproject** файла проекта.
- Скомпилировать и открыть проект в редакторе и настроить необходимые значения
    - Адрес сервера с настроенной Siren
    - Имя (ключ) проекта для идентификации его в джире
    - Префикс файлов логов, которые необходимо прикладывать к репорту
        - Чаще всего он совпадает с именем вашего проекта



Описание классов плагина Krista Bugreport

- **UBugreporterSubsystem**
    - Класс подсистемы баг репортера. Он служит как основная точка доступа при обращении к
        - **UContextRegistry** (основной класс для работы с контекстами)
        - и **UReportController** (точка доступа для создания репорт-сессии и отправки репорта)
- **USentReportChecker**
    - Класс, отвечающий за периодическую проверку отправленных репортов и возвращение информации о созданном в джире тикете

Папки модуля KristaBugreport:

- **Contexts/**
    - Содержит файлы связанные с системой контекстов. Система контекстов подробнее описана ниже.
- **Helpers/**
    - Содержат вспомогательные классы.
    - **UContextPathsHelper**
        - Помогает баг репортеру обработать пути к файлам и папкам: добавить их содержимое в папку репорта.
    - **FJsonHelpers**
        - Содержит ряд вспомогательных статических методов для обработки Json-данных
- **Report/**
    - Основные сущности для формирования и отправки репорта
    - **UBugReport**
        - Главный класс репорта.
        - Здесь расположен функционал для создания скриншотов и логов, сбора данных для репорта в одной папке, создания из нее архива и отправки на сервер.
        - Именно с этим классом работает каждый из реализованных механизмов для обработки запроса пользователя и отправки репорта на сервер.
    - **FReportData**
        - Класс, которому **UBugReport** делегирует большую часть манипуляций с данными
        - Помимо прочего, данные в **FReportData** хранятся в структурах трех типов:
            - **FGameDefinedData**
                - это структура, используемая для сбора и хранения данных о системе, которые репортер может собрать самостоятельно при открытии.

            - Тут должна находиться информация об операционной системе, языке, магазине приложений и т.п.
        - структура **FPlayerDefinedData**
            - предназначена в основном для тех данных, которые репорт получает из пользовательского ввода.
            - Например, это описание проблемы, Email, Subject, ProblemDescription и т.п.
        - **FReportParameters**
            - Это структура, по которой создается в дальнейшем файл **parameters.json**, содержащий настройки для создаваемого в Jira тикета.
            - **FReportParameters** заполняется данными из **FGameDefinedData** и **FPlayerDefinedData**.
    - **FReportScreenshot**, **FZipArchiver**, и другие
        - Вспомогательные классы для отделения функционала по созданию скриншотов, архивированию папок на диске и отправке архива репорта на сервер.
- **UniversalUI/**
    - Папка содержит основное решение рекомендуемое для взаимодействия с багрепортером
    - **UReportController**
        - Тип объекта, который можно получить у **UBugreporterSubsystem** для работы с репортом
        - Через этот объект можно создать новую сессию отправки репорта и отменить или подтвердить его отправку
        - **OpenReport**()
            - Открывает сессию и позволяет получить объект типа **UReportSession**.
        - **SendReport**() и **CancelReport**()
            - Соответственно инициирует отправку, или отменяет отправку репорта.
            - В случае отправки используется последний созданный экземпляр **UReportSession**.
    - **UReportSession** (и конкретная имплементация **UReportSessionImplementation**)
        - Олицетворяет сессию отправки репорта. Тут содержатся и экземпляр **UBugReport** и собранные контекстные данные в виде **FSummaryContextData**.
        - **GetSummaryContextData**()
            - Получить **FSummaryContextData**, которые собрал и записал в сессию при создании **UReportController**.
        - **SetPlayerDefinedData**()
            - Метод для записи данных, собранных у пользователя

    - (введенных через тот интерфейс багрепортера, который там так или иначе реализован на стороне игры)
- Этот метод должен вызываться после открытия окна багрепортера, но перед отправкой репорта методом **UReportController.SendReport**().

Реестр игровых контекстов

Обзор

Реестр игровых контекстов не зависит от подсистемы баг репортера и может использоваться отдельно от него. Багрепортер опрашивает эту систему при создании репортов, чтобы собрать больше информации об игровых системах и добавить ее к отчету.

Идея контекстов состоит в том, чтобы давать произвольным сущностям в игре регистрировать собственные контексты - источники данных о состоянии игры. Пока существует GameInstance, можно обратиться к реестру контекстов и добавить контекст с указанием его уникального имени, а также удалить существующий контекст по его имени. В любой момент времени, у реестра можно запросить данные о текущих контекстах. Каждый зарегистрированный в момент запроса контекст получает возможность добавить свои данные в общий пакет данных контекстов.

Данные контекстов могут быть представлены либо в виде Json-объектов, либо в виде путей к файлам или папкам. Система багрепортера обрабатывает эти данные следующим образом

- Все добавленные контекстами Json-объекты собираются в один общий Json файл с сохранением информации, какой контекст добавил каждый набор Json-данных
    - Если контекст добавил только один Json-объект, то этот объект будет вложен в общий файл с именем соответствующим имени контекста.
    - Если контекст добавил несколько Json-объектов, то будет создан один общий Json-объект с именем контекста, куда будут помещены все добавленные им подобъекты.
- Для добавленных в данные путей, файлы по этим путям будут добавлены в приложенные к репорту файлы
    - Если путь ведет к папке, то папка будет заархивирована и добавлена в репорт в таком виде.
- Также для удобства взаимодействия с системой багрепорта в контексты есть возможность добавить дополнительные пары ключ-значение, которые будут использованы при создании запроса к Siren. Это один из способов добавления значений в Jira-тикеты, генерируемые Siren.

Добавление и удаление контекста

Для создания собственного контекста, данные из которого автоматически попадут в репорт (и следовательно в тикет в Jira), нужно

- Предварительно унаследовать свой класс от базового класса для контекстов
- Обратиться к подсистеме багрепорта для получения реестра контекстов
- Вызвать метод реестра для создания экземпляра контекста по классу и уникального имени
- При необходимости снабдить созданный экземпляр дополнительными данными, необходимыми для его работы

Для удаления контекста достаточно

- Получить реестр контекстов
- Вызвать метод удаления контекста по его имени

API системы контекстов

- **UContextRegistry**
    - Класс реестра контекстов. Он содержит методы для создания и регистрации, а также удаления, собственных контекстов, способных по запросу отдавать данные.
    - **CreateContext**()
        - Метод создает экземпляр контекста по классу и уникальному имени, а также сразу регистрирует его в реестре
        - Класс контекста должен быть унаследован от **UBaseContext**
    - **RemoveContext**()
        - Удаляет зарегистрированный контекст по имени
    - **CollectData**()
        - Собирает данные со всех зарегистрированных контекстов в объект типа **FSummaryContextData**.
- ***ContextData***
    - Существует два класса-интерфейса для данных контекстов
        - **FContextDataContainer**
            - Содержит методы для добавления Json-объектов, а также путей к файлам или папкам. Он используется для заполнения данных со стороны контекстов.
            - Если контексту требуется добавить ровно один Json-объект, следует воспользоваться методом **SetJsonObject**(). Именем для данного объекта станет имя контекста.
            - Если необходимо добавить несколько Json-объектов, используется **AddJsonObject**(), который требует индивидуальное имя для каждого добавляемого объекта.

                - Для автоматического создания Json-объекта по структуре существует метод **AddStruct**(). Он работает как **AddJsonObject**(), но принимает структуру вместо FJsonObject и автоматически создает Json-объект уже внутри метода.
                    - Важно, чтобы структура была помечена макросом **USTRUCT**(), а поля, которые должны попасть в результирующий Json, - **UPROPERTY**().
                - Для добавления пути к файлу или папке используется метод **AddPath**()
            - **FContextDataSource**
                - Служит для чтения собранных данных после получения с помощью метода *UContextRegistry.CollectData*()
                - Здесь есть методы **GetName**(), **GetPaths**(), **GetJsonObject**().
        - **FContextDataImplementation**
            - Этот класс реализует оба вышеописанных интерфейса и используется для собственно хранения данных, полученных от контекстов.
- **FSummaryContextData**
    - Содержит массив данных типа **FContextDataSource** и методы для удобного нахождения данных в этом массиве.
    - Объект этого типа заполняет **UContextRegistry** при запросе **CollectData**().
- **UBaseContext**
    - Класс, от которого должен наследоваться каждый конкретный контекст. Таким образом, в частности, контекст должен быть UObject'ом.
    - Потомки **UBaseContext** должны реализовать метод **GetData**() для добавления своих данных в объект типа **FContextDataContainer**.
    - Кроме этого реализация контекста никак не ограничена.
        - Возможны как случаи, когда все необходимые данные попадают в контекст при его создании,
        - так и реализации, которые собирают данные в момент вызова **GetData**().
        - Контекст может и вовсе не добавлять данных в контейнер, в зависимости от каких-то условий.

Создание игровых контекстов

Для создания нового контекста:

- Прежде всего необходимо унаследовать свой класс от **UBaseContext** и реализовать метод **GetData**() передачи данных в объект типа **FContextDataContainer**.
    - Реализовать логику добавления Json-данных или путей к файлам в теле метода.

- В рантайме, в тот момент, когда данный контекст становится актуальным, необходимо получить экземпляр реестра контекстов, вызвав один из методов **GetContextRegistry**() у подсистемы **UBugreporterSubsystem**.
- У реестра контекстов необходимо вызвать метод **CreateContext<***TMyContextType***>**(*Name*).
    - Важно передать уникальное имя для данного контекста.
    - Если будет совершена попытка зарегистрировать два контекста с одним именем, сработает *ensure* и второй контекст добавлен не будет.
- Метод **CreateContext**() возвращает созданный экземпляр контекста указанного типа.
    - Если данный контекст требует инициализации (например, инъекции каких-то зависимостей), следует воспользоваться возвращенным таким образом объектом.
- В момент, когда контекст потеряет актуальность его можно удалить по уникальному имени, указанному при создании
    - Для этого, как и при создании, нужно получить объект реестра
    - и вызвать у него метод **RemoveContext**().

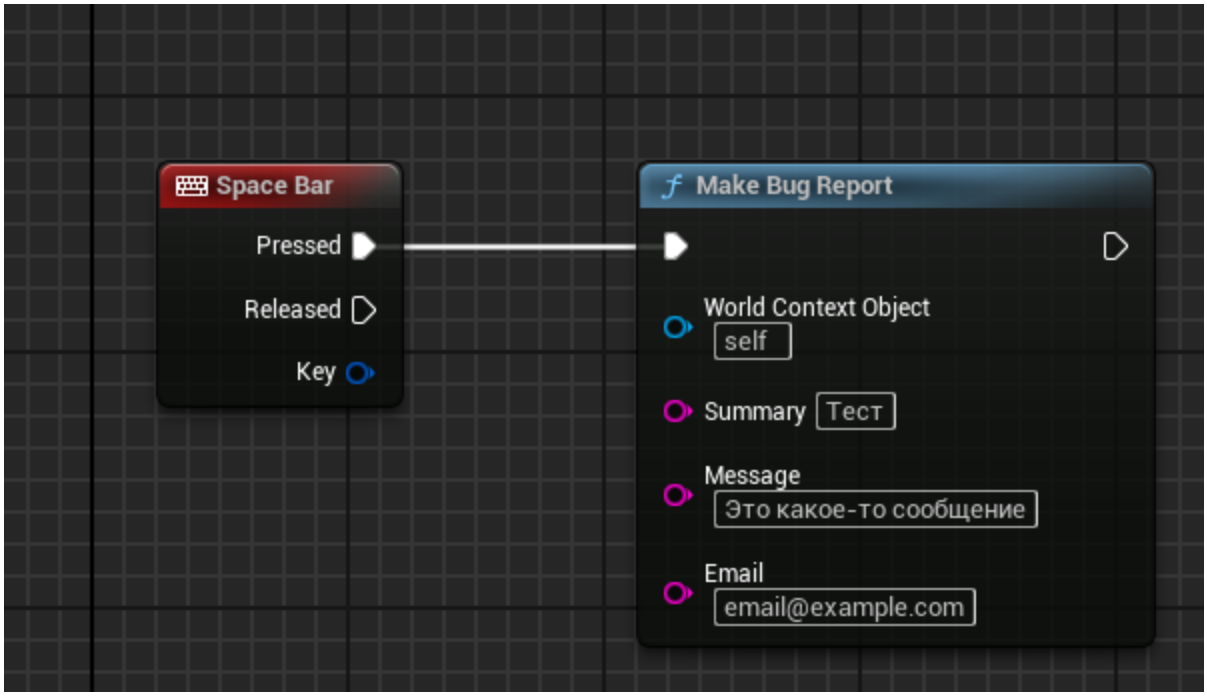
Примеры контекстов:

- **KristaBugreport/Contexts/Examples**
    - В этой папке можно найти несколько примеров контекстов, возвращающих данные разным способом
    - **UStructDataContext**
        - Демонстрирует простоту создания контекста, передающего данные в виде структуры
    - **ULogsPathContext**
        - Демонстрирует добавление к контекстным данным путей к файлам и папкам
    - **UJsonWriterContext**
        - Пример создания Json-объекта на лету с помощью **TJsonWriter**'а

Проверка работоспособности плагина

Действия совершаются в новом пустом проекте Unreal (Empty Project - C++) после установки плагина с зависимостями. Проверка не требует настройки сервиса Siren, поскольку проверяется только локальное создание репорта.

- Открыть блюпринт уровня и собрать следующий граф



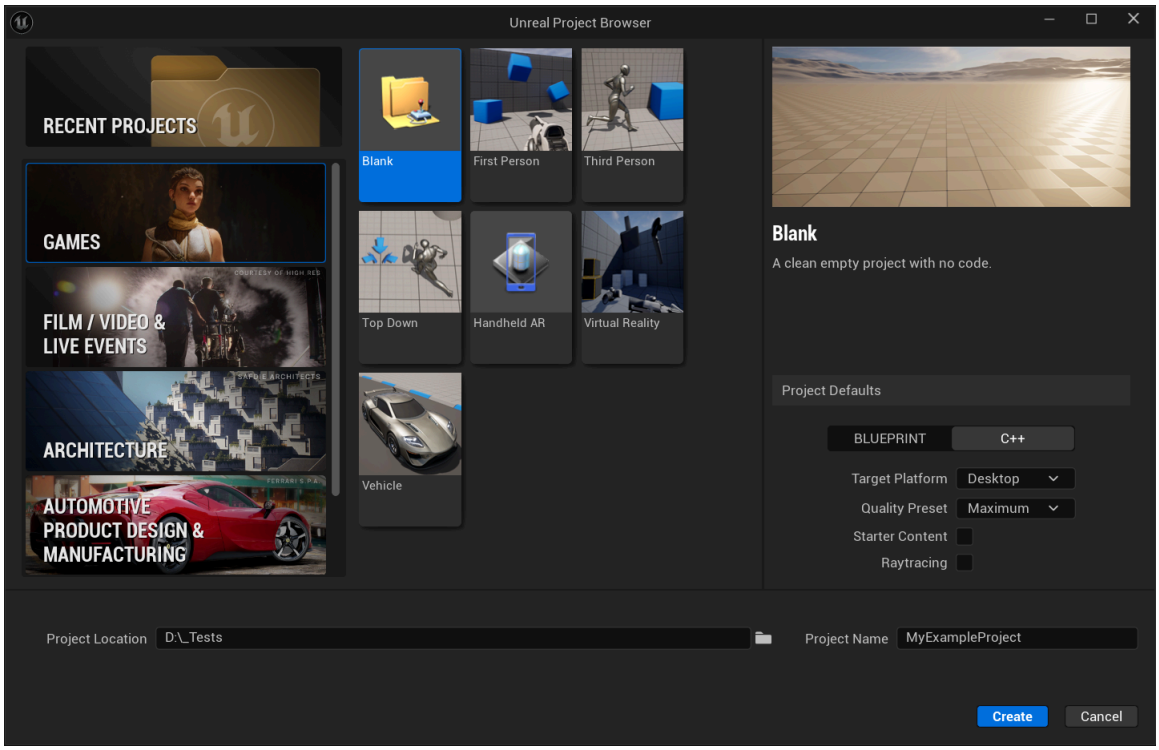
- Запустить игру
- Запустить игру на уровне, кликнуть на игровой области чтобы гарантировать положение фокуса ввода.
- Нажать пробел
- Открыть в Проводнике папку Saved\BugIt\WindowsEditor в проекте (например C:\MyProject\Saved\BugIt\WindowsEditor)
- Удостовериться что в ней появился архив с именем вида Report__20250113_132058_Compressed.zip
- Открыть архив, удостовериться что в нем есть файл parameters.json, содержащий отправленные данные:


```
{
    "Guid": "37a4b8d7-45df-e358-26a2-f3bb6d68e80e",
    "Project": "",
    "ReportDateTime": "2025.01.13-11.36.41",
    "Email": "email@example.com",
    "Priority": "Normal",
    "Summary": "Тест",
    "CustomMessage": "Это какое-то сообщение"
}
```

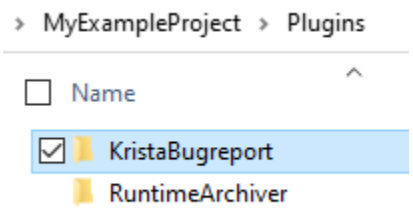
Создание простого репорта на стороне игры

Пройдём минимальные шаги для создания проекта, который способен формировать и отправлять репорты с помощью плагина Krista Bugreport

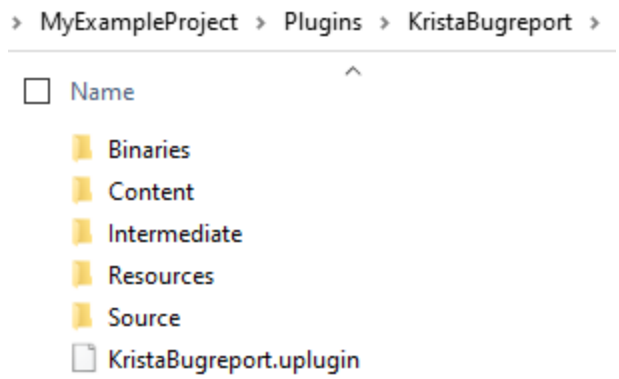
1. Создать проект с названием **MyExampleProject**
    a. Выбираем в настройках **Project Defaults** опцию **C++**

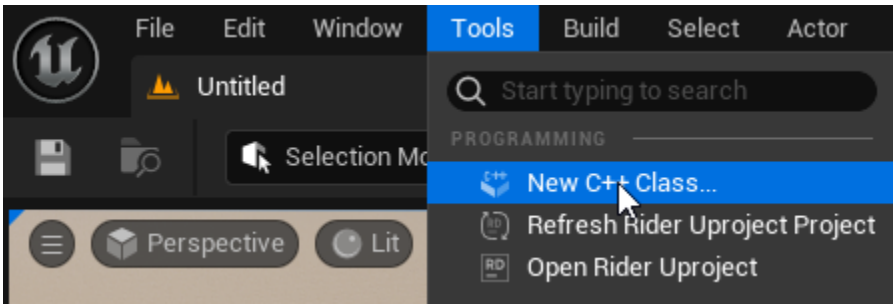


2. Закрываем Unreal Editor и открываем папку проекта. Внутри папки проекта необходимо создать папку **Plugins**, если она не существует.
3. Поместить папки плагинов **KristaBugreport** и **RuntimeArchiver** в папку Plugins проекта
    a. Должен получиться путь *MyExampleProject/Plugins/KristaBugreport/*

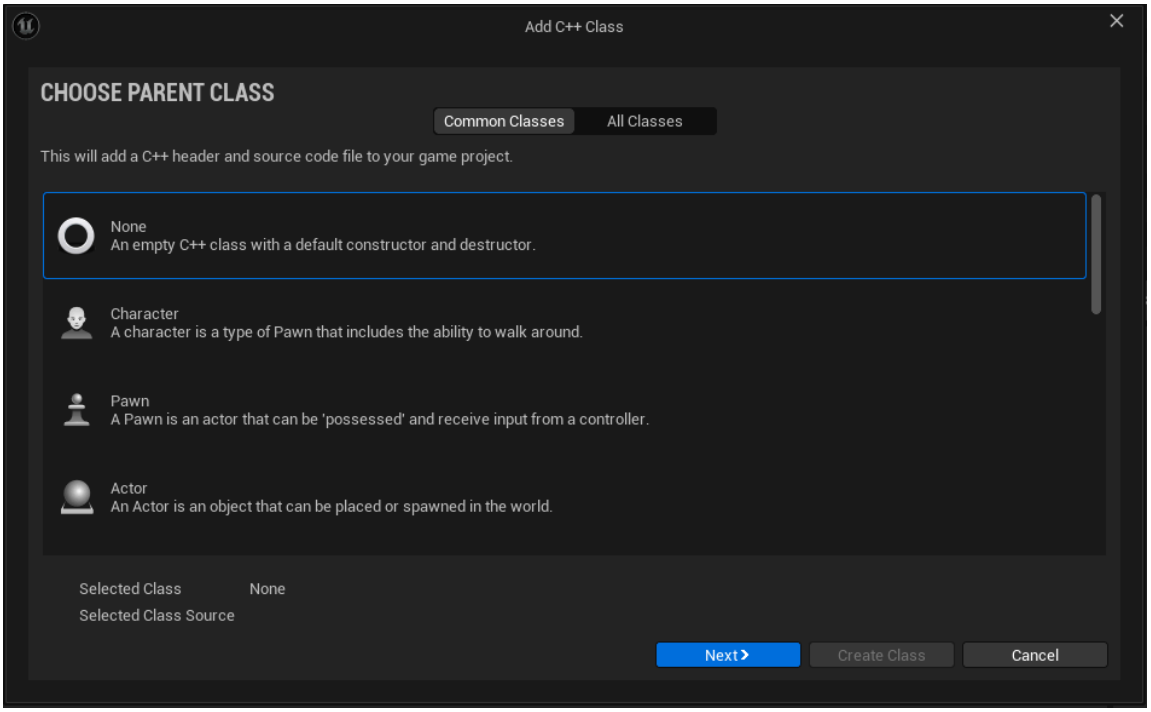


    b.

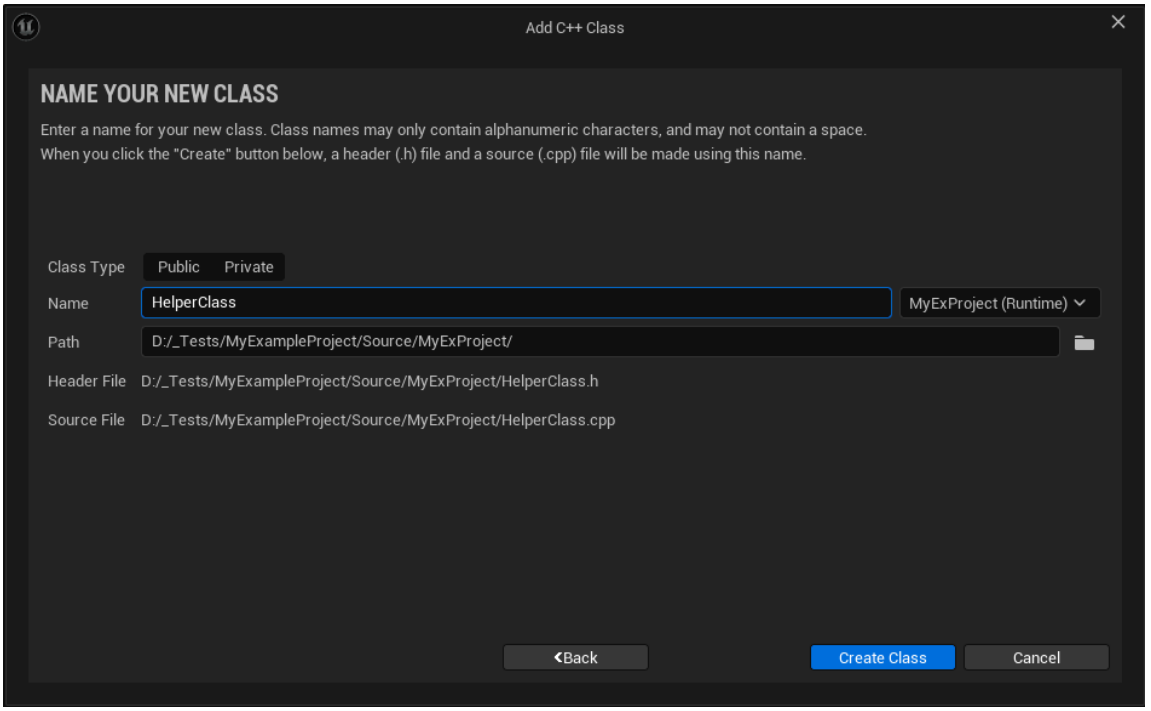
c. 

4. Закрыть редактор кода, если он открылся при создании проекта.
5. Если проект изначально создан как blueprint-проект и не содержит кода, для следующего шага необходимо сделать его проектом с кодом.
    a. Открываем проект в Unreal Editor
    b. Переходим в меня Tools > New C++ Class
        i. 

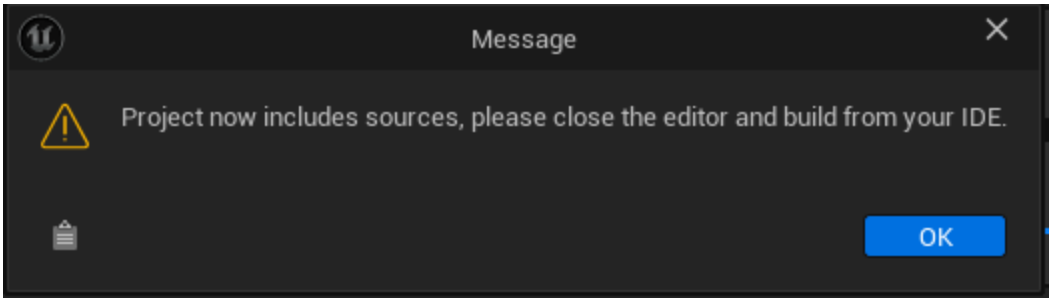
    c. В открывшемся окне ничего не меняем и нажимаем Next

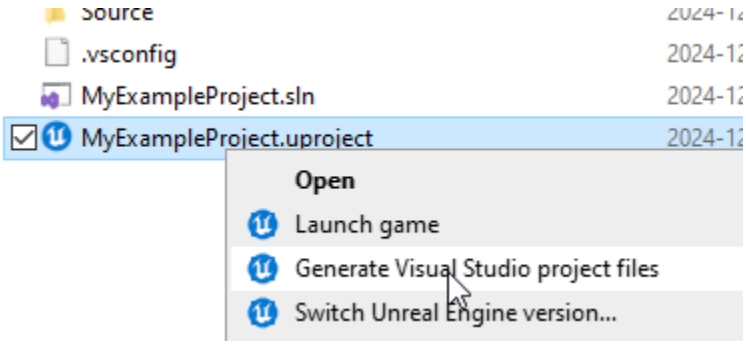
i. 

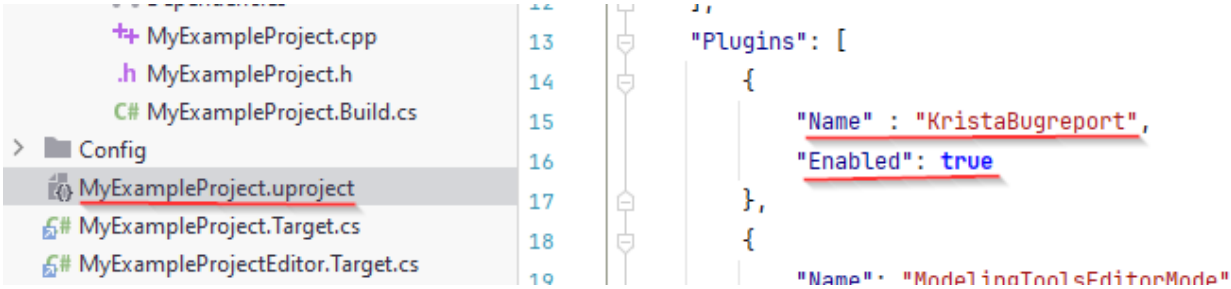
d. В следующем окне вводим желаемое имя класса, или оставляем его как есть. Нажимаем Create Class

i. 

e. Теперь мы можем работать с проектом в IDE. Закрываем редактор Unreal.

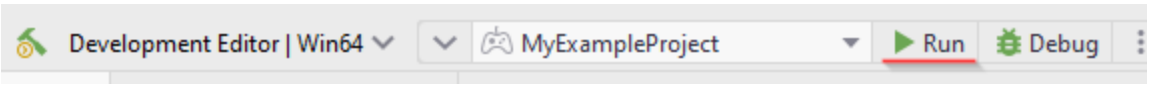
i. 

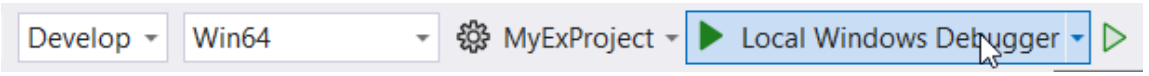
6. Генерируем (или перегенерируем, если он уже существует) файл решения
   a. 

7. Затем необходимо открыть проект в IDE
8. Прописать плагин в **.uporject** файла вашего проекта
   a. 

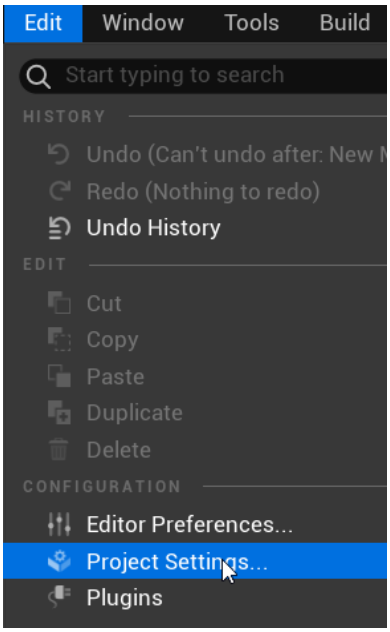
9. Добавить в **.Build.cs** файл того модуля, из которого будет использоваться плагин, зависимость от добавленного плагина
   a. 

10. На этом этапе имеет смысле скомпилировать и запустить проект в редакторе, чтобы убедиться, что плагин успешно добавлен.
    a. Если вы работаете в Rider, запускам проект с помощью кнопки Run

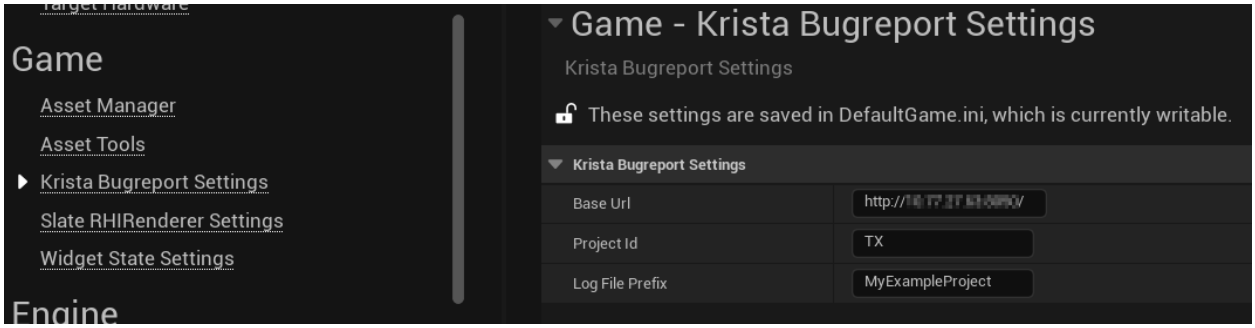
i. 

b. Если вы работаете в Visual Studio может потребоваться в первый раз выбрать ваш проект как основной
   i. Для этого кликаем правой кнопкой на имени проекта в Solution Explorer и выбираем опцию Set as Startup Project
   ii. Теперь проект можно запустить
iii. 

11. Открыть меню **Edit > Project Settings**

a. 

12. Там в разделе Game > Krista Bugreport Settings вводим настройки для своего проекта
    a. Адрес сервера с настроенной Siren
    b. Имя (ключ) проекта для идентификации его в Jira
    c. Префикс файлов логов, которые необходимо прикладывать к репорту
       i. Обычно он совпадает с именем вашего проекта



d. *Замечание*: Если раздел *Game* > *Krista Bugreport Settings* у вас в Project Settings отсутствует, значит установка плагина не прошла успешно. Перепроверьте предыдущие шаги.

13. Закрываем Unreal Editor
14. Далее необходимо добавить код для создания и отправки репорта. В минимальном сценарии достаточно:
    a. Обратиться к контроллеру и запросить сессию
    b. Каким-то образом получить и заполнить player-data
    c. Обратиться к контроллеру и запросить отправку
15. Пройдем эти шаги на простом примере. Для удобства создадим подсистему UGameBugreportSubsystem (класс, унаследованный от UGameInstanceSubsystem), которая будет обрабатывать ввод от пользователя и отправлять репорты.

```
UCLASS()
No derived blueprint classes
class PLUGINSTESTSPROJECT_API UGameBugreportSubsystem : public UGameInstanceSubsystem
{
    GENERATED_BODY()
};
```

16. Добавим код для подписки на сочетание клавиш для открытия репорта - Alt+B
    a. Создаём класс `UGameBugreportSubsystem` и добавляем в GameBugreportSubsystem.h код

```
//-- GameBugreportSubsystem.h
#include "CoreMinimal.h"
#include "Subsystems/GameInstanceSubsystem.h"
#include "GameBugreportSubsystem.generated.h"

class UReportSession;

UCLASS()
class MYEXAMPLEPROJECT_API UGameBugreportSubsystem : public
UGameInstanceSubsystem
```

```
{
	GENERATED_BODY()

public:
	virtual  void  Initialize(FSubsystemCollectionBase&  Collection)
override;
	void OnLocalPlayerAdded(ULocalPlayer* LocalPlayer);
private:
	bool bKey1Down;
	bool bKey2Down;

	FOverrideInputKeyHandler PreviousOverrideDelegate;
	bool OnOverrideInputKey(FInputKeyEventArgs& InputKeyEventArgs);

	void SendReport();

};
```

b. Добавляем в GameBugreportSubsystem.cpp код

```
//-- GameBugreportSubsystem.cpp
#include "GameBugreportSubsystem.h"
#include "BugreporterSubsystem.h"
#include "UniversalUI/ReportController.h"

void    UGameBugreportSubsystem::Initialize(FSubsystemCollectionBase&
Collection)
{
	Super::Initialize(Collection);
	GetGameInstance()->OnLocalPlayerAddedEvent.AddUObject(this,
&ThisClass::OnLocalPlayerAdded);
}

void        UGameBugreportSubsystem::OnLocalPlayerAdded(ULocalPlayer*
LocalPlayer)
{
	if              (auto*              ViewportClient              =
LocalPlayer->GetGameInstance()->GetGameViewportClient())
	{
		PreviousOverrideDelegate                                    =
ViewportClient->OnOverrideInputKey();
		ViewportClient->OnOverrideInputKey().BindUObject(this,
&ThisClass::OnOverrideInputKey);
	}
}
```

```
bool  UGameBugreportSubsystem::OnOverrideInputKey(FInputKeyEventArgs&
InputKeyEventArgs)
{
	if (InputKeyEventArgs.Key == EKeys::B)
	{
		if (InputKeyEventArgs.Event == IE_Pressed)
		{
			bKey1Down = true;
		}
		if (InputKeyEventArgs.Event == IE_Released)
		{
			bKey1Down = false;
		}

		if (bKey1Down && bKey2Down)
		{
			SendReport();
			return true;
		}
	}

	if	(InputKeyEventArgs.Key	==	EKeys::LeftAlt	||
InputKeyEventArgs.Key == EKeys::RightAlt)
	{
		if (InputKeyEventArgs.Event == IE_Pressed)
		{
			bKey2Down = true;
		}
		if (InputKeyEventArgs.Event == IE_Released)
		{
			bKey2Down = false;
		}

		if (bKey1Down && bKey2Down)
		{
			SendReport();
			return true;
		}
	}

	if (PreviousOverrideDelegate.IsBound())
	{
		return
PreviousOverrideDelegate.Execute(InputKeyEventArgs);
	}
```

```
	return false;
}
```

17. Далее нужно добавить код, который собственно сформирует и отправит репорт на сервер. Закрываем редактор Unreal и возвращаемся в IDE.
18. Необходимо создать три метода
```
	void SendReport();
	void OnSessionInitialized(UReportSession* ReportSession);
	void OnReportIsSent(bool bSuccess);
```

19. Прежде всего метод **SendReport()**, который был ранее привязан к сочетанию клавиш. Тут нам необходимо создать сессию репорта

```
void UGameBugreportSubsystem::SendReport()
{
	UReportController* ReportController =
UBugreporterSubsystem::GetReportController(this);

	if (!ReportController)
	{
		return;
	}

	if (ReportController->IsOpenReport())
	{
		ReportController->CancelReport();
		return;
	}

	FReportSessionReadyDelegate Callback;
	Callback.BindUObject(this, &ThisClass::OnSessionInitialized);
	ReportController->OpenReport(Callback);
}
```

20. Метод **OnSessionInitialized()** будет вызван, когда **KristaBugreport** проведет подготовку данных репорта, включая сбор данных от контекстов.
    a. Здесь можно открыть виджет багрепортера, созданный на стороне игры, и получить пользовательский ввод. Для примера мы ограничимся захардкоженными в тексте значениями.
    b. Для отправки репорта необходимо вызвать метод **SendReport()** у контроллера репорта
        i. `ReportController->SendReport(Callback);`

c. Если отправку требуется отменить из-за пользовательского ввода, или ошибки, необходимо закрыть сессию вызовом **CancelReport()**
    i. `ReportController->CancelReport();`

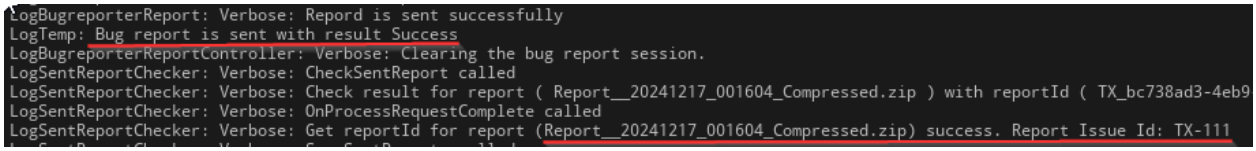
```
void UGameBugreportSubsystem::OnSessionInitialized(UReportSession*
ReportSession)
{
	if (!ReportSession)
	{
		UE_LOG(LogTemp, Error, TEXT("Failed to open a bug report
session!"))
		return;
	}

	UReportController* ReportController =
UBugreporterSubsystem::GetReportController(this);

	if (!ensureAlwaysMsgf(ReportController && ReportSession,
TEXT("Report controller or session is invalid!")))
	{
		if (ReportController)
		{
			ReportController->CancelReport();
		}
		return;
	}

	//-- Основная часть метода - наполнение репорта
пользовательскими данными --
	FPlayerDefinedData PlayerData;
	PlayerData.Email = "My@email.com";
	//-- Описание бага --
	PlayerData.Summary = "This is a new bug!";
	PlayerData.CustomMessage = "Steps to reproduce this bug are:
\nFirst\nSecond\nThird";
	//-- Добавим несколько дополнительных полей для Jira --
	PlayerData.CustomParameters.Add("Assignee", TEXT("Ivanov"));
	PlayerData.CustomParameters.Add("QA", TEXT("Petrov"));
	PlayerData.CustomParameters.Add("FixVersion", TEXT("TEST:
Update 1.0.1"));
	//-- Запишем введённые данные в репорт --
	ReportSession->SetPlayerDefinedData(PlayerData);

	FReportControllerSendReportCallback Callback;
	Callback.BindUObject(this, &ThisClass::OnReportIsSent);
```

```
	ReportController->SendReport(Callback);
}
```

21. Наконец в подписанном ранее методе **OnReportSent()** можно обработать результат отправки репорта.

```
void UGameBugreportSubsystem::OnReportIsSent(bool bSuccess)
{
	if (bSuccess)
	{
		UE_LOG(LogTemp, Verbose, TEXT("Bug report is sent with
result Success"));
	}
	else
	{
		UE_LOG(LogTemp, Warning, TEXT("Bug report is sent with
result Failure"));
	}
}
```

22. Компилируем проект и запускаем его в Unreal Editor.
23. Запускаем плеймод (**Alt+P**).
24. Теперь нажатие **Alt-B** будет приводить к вызову созданного ранее метода **SendReport()** и, соответственно отправке наших данных на сервер.
25. Информацию об отправленном репорте можно увидеть в окне Output Log.
    a. Если ваша Siren настроена на взаимодействие с Jira и все необходимые поля Jira в вашем запросе присутствуют, в логе вы также сможете увидеть полученный от сервера номер тикета.



26. Ниже приведён полный листинг класса созданного на стороне игры для взаимодействия с багрепортом:

//-- **GameBugreportSubsystem.h** --

```
#pragma once

#include "CoreMinimal.h"
```

```
#include "Subsystems/GameInstanceSubsystem.h"
#include "GameBugreportSubsystem.generated.h"

class UReportSession;

UCLASS()
class MYEXAMPLEPROJECT_API UGameBugreportSubsystem : public
UGameInstanceSubsystem
{
	GENERATED_BODY()

public:
	virtual void Initialize(FSubsystemCollectionBase& Collection) override;
	void OnLocalPlayerAdded(ULocalPlayer* LocalPlayer);
private:
	bool bKey1Down;
	bool bKey2Down;

	FOverrideInputKeyHandler PreviousOverrideDelegate;
	bool OnOverrideInputKey(FInputKeyEventArgs& InputKeyEventArgs);

	void SendReport();
	void OnSessionInitialized(UReportSession* ReportSession);
	void OnReportIsSent(bool bSuccess);

};
```

//-- **GameBugreportSubsystem.cpp** --

```
#include "GameBugreportSubsystem.h"

#include "BugreporterSubsystem.h"
#include "UniversalUI/ReportController.h"

void UGameBugreportSubsystem::Initialize(FSubsystemCollectionBase& Collection)
{
	Super::Initialize(Collection);
	GetGameInstance()->OnLocalPlayerAddedEvent.AddUObject(this,
&ThisClass::OnLocalPlayerAdded);
}
```

```
void UGameBugreportSubsystem::OnLocalPlayerAdded(ULocalPlayer* LocalPlayer)
{
	if (auto* ViewportClient = LocalPlayer->GetGameInstance()->GetGameViewportClient())
	{
		PreviousOverrideDelegate = ViewportClient->OnOverrideInputKey();
		ViewportClient->OnOverrideInputKey().BindUObject(this, &ThisClass::OnOverrideInputKey);
	}
}

bool UGameBugreportSubsystem::OnOverrideInputKey(FInputKeyEventArgs& InputKeyEventArgs)
{
	if (InputKeyEventArgs.Key == EKeys::B)
	{
		if (InputKeyEventArgs.Event == IE_Pressed)
		{
			bKey1Down = true;
		}
		if (InputKeyEventArgs.Event == IE_Released)
		{
			bKey1Down = false;
		}

		if (bKey1Down && bKey2Down)
		{
			SendReport();
			return true;
		}
	}

	if (InputKeyEventArgs.Key == EKeys::LeftAlt || InputKeyEventArgs.Key == EKeys::RightAlt)
	{
		if (InputKeyEventArgs.Event == IE_Pressed)
		{
			bKey2Down = true;
		}
		if (InputKeyEventArgs.Event == IE_Released)
		{
```

```
                bKey2Down = false;
            }

            if (bKey1Down && bKey2Down)
            {
                SendReport();
                return true;
            }
        }

        if (PreviousOverrideDelegate.IsBound())
        {
            return PreviousOverrideDelegate.Execute(InputKeyEventArgs);
        }

        return false;
}

void UGameBugreportSubsystem::SendReport()
{
        UReportController* ReportController =
UBugreporterSubsystem::GetReportController(this);

        if (!ReportController)
        {
            return;
        }

        if (ReportController->IsOpenReport())
        {
            ReportController->CancelReport();
            return;
        }

        FReportSessionReadyDelegate Callback;
        Callback.BindUObject(this, &ThisClass::OnSessionInitialized);
        ReportController->OpenReport(Callback);
}

void UGameBugreportSubsystem::OnSessionInitialized(UReportSession*
ReportSession)
{
```

```
	if (!ReportSession)
	{
		UE_LOG(LogTemp, Error, TEXT("Failed to open a bug report session!"))
		return;
	}

	UReportController* ReportController = UBugreporterSubsystem::GetReportController(this);

	if (!ensureAlwaysMsgf(ReportController && ReportSession, TEXT("Report controller or session is invalid!")))
	{
		if (ReportController)
		{
			ReportController->CancelReport();
		}
		return;
	}

	//-- Основная часть метода - наполнение репорта пользовательскими данными --
	FPlayerDefinedData PlayerData;
	PlayerData.Email = "My@email.com";
	//-- Описание бага --
	PlayerData.Summary = "This is a new bug!";
	PlayerData.CustomMessage = "Steps to reproduce this bug are: \nFirst\nSecond\nThird";
	//-- Добавим несколько дополнительных полей для Jira --
	PlayerData.CustomParameters.Add("Assignee", TEXT("Ivanov"));
	PlayerData.CustomParameters.Add("QA", TEXT("Petrov"));
	PlayerData.CustomParameters.Add("FixVersion", TEXT("TEST: Update 1.0.1"));
	//-- Запишем введённые данные в репорт --
	ReportSession->SetPlayerDefinedData(PlayerData);

	FReportControllerSendReportCallback Callback;
	Callback.BindUObject(this, &ThisClass::OnReportIsSent);
	ReportController->SendReport(Callback);
}

void UGameBugreportSubsystem::OnReportIsSent(bool bSuccess)
{
	if (bSuccess)
```

```
	{
		UE_LOG(LogTemp, Log, TEXT("Bug report is sent with result Success"));
	}
	else
	{
		UE_LOG(LogTemp, Log, TEXT("Bug report is sent with result Failure"));
	}
}
```

Siren: Автоматизация обработки баг-репортов с интеграцией с Jira

Siren — это сервис, предназначенный для автоматизации процесса обработки баг-репортов и их интеграции с системой управления проектами **Jira**.

Ключевые возможности:

1. **Автоматическое создание задач:**
   Siren принимает баг-репорты через HTTP-запросы, анализирует их содержимое и создает задачи в Jira с полным набором данных: описанием проблемы, вложениями и метаданными.
2. **Аналитика и статистика:**
   Сервис генерирует статистические графики и отчеты по каждому проекту, предоставляя детализированную информацию о статусе и динамике обработки багов.
3. **Простота интеграции:**
   Siren легко настраивается и интегрируется с любым проектом, использующим Jira.

Как работает Siren?

1. Баг-репорт отправляется по указанному HTTP-адресу.
2. Содержимое репорта автоматически преобразуется в задачу в Jira.
3. Пользователь получает доступ к задачам и аналитике в интуитивно понятном интерфейсе.

Преимущества Siren:

- **Ускорение рабочих процессов:** Полная автоматизация заведений задач.
- **Интеграция с Jira:** Быстрая настройка без необходимости сложной конфигурации.
- **Детальная аналитика:** Графики и статистика помогают контролировать ход работы над проектом.

Подготовка к установке

Перед установкой сервиса Siren необходимо установить Jira (версии от 8.0.0 и выше)
https://confluence.atlassian.com/adminjiraserver/installing-jira-applications-938846823.html

Установка Siren

Siren на текущий момент поддержка установку только Debian/Ubuntu.

Установка на платформе Debian/Ubuntu

1. Установка пакета lsb-release, если он не установлен, для определение версии в командах:
    a. sudo apt-get -y install lsb-release
2. Установка СУБД Postgres командами:
    a. sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs 2>/dev/null)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
    b. wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
    c. sudo apt-get update && sudo apt-get -y install postgresql
3. Настройка Postgres командами:

1. sudo su postgres
2. psql
3. \password <Ваш пароль>
4. \q
5. exit

4. Установка среды выполнения Net9 командами:
    a. wget https://packages.microsoft.com/config/debian/$(lsb_release -sr 2>/dev/null)/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
    b. sudo dpkg -i packages-microsoft-prod.deb && rm packages-microsoft-prod.deb
    c. sudo apt-get update && apt-get install -y dotnet-sdk-9.0
5. Установка сервиса командой:
    a. dpkg -i siren-server_1.0.0_all.deb
6. Сервис установлен и можно переходить к первоначальной настройке

Запуск сервиса осуществляется командой systemctl start siren-server.service
Перезапуск сервиса осуществляется командой systemctl restart siren-server.service
Проверка статуса сервиса осуществляется командой systemctl status siren-server.service

Конфигурация

Конфигурация сервиса хранится в файле
Debian: /etc/siren-server/appsettings.json

Все нижеуказанные настройки выполняются в нем.

Первоначальная конфигурация

После установки в первую очередь необходимо настроить соединение с базой данных и jira. Для это нужно в файле настроек выполнить следующее:

1. В разделе ConnectionStrings в подразделе Default указать полную строку подключения в виде “*server=localhost;UserId=postgres;Password=пароль указанный при установке Postgresql;database=siren;”*
2. В разделе Jira заполнить поля:
    1. **Host** - адрес до сервера jira
    2. **User** - имя пользователя jira, от которого будет осуществляться вход (Рекомендуется создать отдельную учетную запись)
    3. **Password** - пароль пользователя jira, от которого будет осуществляться вход (Рекомендуется создать отдельную учетную запись)

После этого сервис можно перезапускать и пользоваться.
Сервис будет доступен по адресу http:///<ip адрес сервера>:8050/

Опции конфигурации

Наименование	Описание	Пример
Раздел **General**		
MaxReceiveFileSize	максимальный размер принимаемого файла репорта. По-умолчанию 512мб	"MaxReceiveFileSize": 536870912
CleanupObsoleteDays	период очистки устаревших репортов в дня. По-умолчанию 30. Если указать 0, то старые репорты удаляться не будут	"CleanupObsoleteReports": 30
ReportStoragePath	путь до хранилища репортов. По-умолчанию это /var/lib/siren-server/storage	
Раздел **Jira**		
Host	адрес сервера Jira, к которому будет подключаться сервис. Полный в виде https://[::]	"Host": "http://localhost:8080"
User	имя пользователя jira, от которого будет осуществляться вход (Рекомендуется создать отдельную учетную запись)	"User": "siren"
Password	пароль пользователя jira, от которого	"Password": "1234567"

	будет осуществляться вход (Рекомендуется создать отдельную учетную запись)	

CustomFieldMap	карта пользовательских полей.
	Поле необязательно. Используется для установки значений из parameters.json репорта, если там поле указано с другим именем нежели в Jira. Заполняется в формате:
	Parameter - наименование параметра в файле
	Name - наименование поля Jira,
	Id - идентификатор поля Jira,
	Default - значение поля по-умолчанию.
		"CustomFieldMap": [
		{
		"Parameter": "Parameter01",
		"Name": "Field01",
		"Id": "10001",
		"Default": ""
		}
		]

Отправка и получение репортов:

Репорты отправляются из клиентского плагина для UE5. В случае если нужна отправка другим способом, то

Для получения репорта достаточно отправить zip файл архива на адрес *<базовый адрес Siren>/report* POST запросом.
В запрос кроме файла можно отправить небольшой json в поле parameters с содержимым:

1. **Project** - ключ проекта Jira
2. **Guid** - уникальный идентификатор репорта
3. **Email** - электронная почта отправителя

Архив обязательно должен содержать следующие файлы:

1. **Message.txt** - файл с описанием баг репорта. Это текст будет добавлен в поле Description задачи. Так же первая строка или первые 200 символов будут использованы для формирования Summary задачи.
2. **Parameters.json** - файл с параметрами в виде ключ:значение, для заполнения остальных полей задачи. Ключи не должны повторяться.

Структура файла Parameters.json

Файл параметров должен содержать следующие обязательные поля:

1. **Project** - ключ проекта Jira. Необходим для создания задачи
2. **Guid** - уникальный идентификатор репорта. Необходим для идентификации репорта.

Так же файл репортов имеет зарезервированные необязательные поля:

1. **IssueType** - тип задачи Jira.
2. **Priority** - приоритет задачи Jira
3. **Labels** - список меток задачи Jira. Метки не должны содержать пробелов. Указываются в строчку через точку с запятой. Например, Bugreport;Label1;Label2.
4. **Components** - список компонентов задачи. Указываются в строчку через точку с запятой. Например, Component1;Component System;Component_2
5. **FixVersions** - список версий для исправления задачи. Указываются в строчку через точку с запятой. Например, Project: Version 1;Project: DLC1
6. **Epic** - ключ задачи типа Epic, к которой будет привязана задача по репорту

Остальные поля должны быть либо смаплены через настройки (см. CustomFieldMap в разделе Jira опций конфигурации), либо ключ поля должен соответствовать названию поля в Jira.

Все остальные файлы репорта будут добавлены к задаче как вложения.

Ответ возвращается в виде json с полями:

1. **success** - результат успешности получения репорта.
2. **error** - сообщение об ошибке, если получение репорта не удалось
3. **reportid** - наименование файла, которые было сформировано при получении
4. **original_file_name** - наименование файла, который был отправлен

Метрики

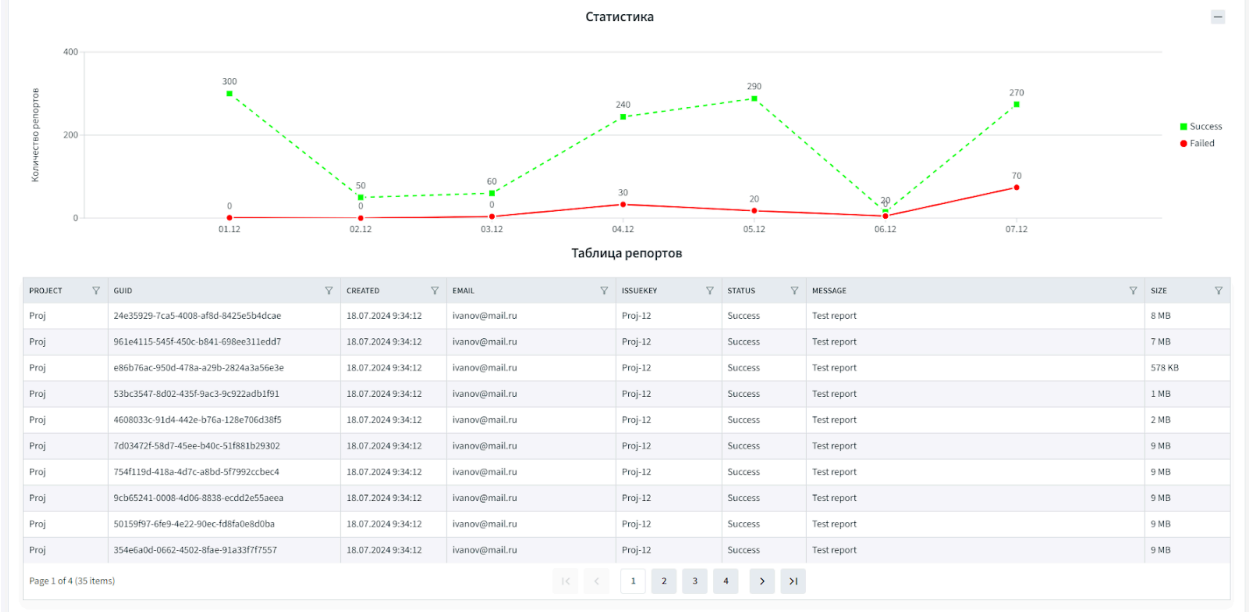
Сбор статистических данных в Siren ведется как через Prometeus, так внутренними системами. Внутренние системы используются для хранения общий данных по процессам приема и обработки репортов, когда Prometeus используется для более детализированных данных.

Список метрик Prometeus

1. **siren_bugreport_received_total** - общее количество полученных репортов
2. **siren_bugreport_receive_errors_total** - количество ошибок получения репортов
3. **siren_reports_processed** - количество репортов по проекту обработано в разных статусах
4. **siren_report_processing_time** - гистограмма времени обработки репортов по проекту (в миллисекундах)
5. **siren_report_processing_time_summary** - сводка времени обработки репортов за 10 мин.

Web Interface

Главное окно



На главном окне расположены вкладки по проектам полученных репортов. Вкладки генерируются автоматически в зависимости от количества проектом. На каждой вкладке расположено 2 виджета:

1. **Статистика** - тут отображаются данные об обработке репортов за последние 7 дней. Можно свернуть.
2. **Таблица репортов** - список репортов по проекту. Двойной клик по репорту позволяет открыть страницу с его детальной информацией

На странице подробностей репорта расположены все данные о репорте и кнопки взаимодействия с ним:

1. **Удалить** - если репорт более не нужен

2. **Повторить** - если при обработке репорта были ошибки. В противном случае кнопка не отображается.

Tiamat: Система управления Git-хуками для платформы GitLab

Tiamat — это система управления хуками Git, разработанная для работы с множеством репозиториев на платформе **GitLab**.

Основные возможности:

1. **Распределенная модель управления:**
   Поддержка как централизованных, так и кластерных инсталляций GitLab для гибкой работы с хуками.
2. **Централизованный контроль:**
   Управление всеми хуками через единый координатор и веб-интерфейс, что обеспечивает удобство мониторинга и настройки.
3. **Настраиваемые правила:**
   Для каждого репозитория можно задать уникальный набор правил, применяемых к каждому коммиту, с детализированным журналированием всех проверок.
4. **Интегрированное логирование:**
   Единое место для получения сообщений и отладочной информации, упрощающее анализ и решение возникающих проблем.

Преимущества Tiamat:

- **Высокая производительность:**
  Использование стека на основе .NET позволяет минимизировать нагрузку на сервер, обеспечивая высокую скорость проверки изменений.
- **Продвинутое кеширование:**
  Оптимизация за счет работы с объектами и сущностями Git на низком уровне делает систему независимой от изменяемого окружения.
- **Гибкость:**
  Подходит для любых конфигураций GitLab, от локальных инсталляций до масштабируемых кластеров.

Как работает Tiamat?

1. Управляйте множеством хуков через централизованный интерфейс.
2. Настраивайте индивидуальные правила для каждого репозитория.
3. Получайте детализированные логи исполнения для контроля над процессом.

Информация о продукте:

- **Поддерживаемые платформы:** GitLab (централизованные и кластерные установки).
- **Архитектура:** Основана на .NET для оптимальной производительности.
- **Доступность:** Совместима с любыми окружениями GitLab.

Tiamat — это надежное решение для управления хуками Git, обеспечивающее стабильность, удобство и полный контроль за изменениями в ваших репозиториях.

Подготовка к установке

Перед установкой Tiamat необходимо установить и настроить его зависимости, если они еще не установлены:

1. Gitlab (https://docs.gitlab.com/omnibus/installation/) - собственно сам сервис управления git репозиториями. Желательно установку Omnibus, версии от 17.3 и выше. На ос Debian
2. Consul (https://www.consul.io/) - сервис централизованной конфигурации, для управления конфигурацией хуков. После установки необходимо сформировать корневой ключ конфигурации Tiamat, например, services/tiamat

Установка Tiamat Hooks Pre-Receiver

Так как хуки запускаются внутренним механизмом **Gitlab**, то их запуска используется отдельный пакет на Debian **tiamat-hooks**. Его необходимо установить на том же сервере, что и сам **Gitlab.**

1. Установка пакета lsb-release, если он не установлен, для определение версии в командах:
    1. sudo apt-get -y install lsb-release
2. Установка среды выполнения Net 9 командами:
    1. wget https://packages.microsoft.com/config/debian/$(lsb_release -sr 2>/dev/null)/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
    2. sudo dpkg -i packages-microsoft-prod.deb && rm packages-microsoft-prod.deb
    3. sudo apt-get update && apt-get install -y dotnet-sdk-9.0
3. Установка сервиса командой:
    1. dpkg -i tiamat-hooks_1.0.0_all.deb
4. Если репозитории хранится не на машине с установленным гитлабом, то нужно предоставить хуким физический доступ для до них.
5. Пакет установлен и можно переходить к первоначальной настройке

Установка Tiamat Manager

Для управления хуками используется отдельный сервис **tiamat-manager**

1. Установка пакета lsb-release, если он не установлен, для определение версии в командах:

    1. sudo apt-get -y install lsb-release
2. Установка СУБД Postgres командами:
    1. sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs 2>/dev/null)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
    2. wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
    3. sudo apt-get update && sudo apt-get -y install postgresql
3. Настройка Postgres командами:
1. sudo su postgres
2. psql
3. \password <Ваш пароль>
4. \q
5. exit
4. Установка среды выполнения Net9 командами:
    1. wget https://packages.microsoft.com/config/debian/$(lsb_release -sr 2>/dev/null)/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
    2. sudo dpkg -i packages-microsoft-prod.deb && rm packages-microsoft-prod.deb
    3. sudo apt-get update && apt-get install -y dotnet-sdk-9.0
5. Установка сервиса командой:
    1. dpkg -i tiamat-manager_1.0.0_all.deb

   Сервис установлен и можно переходить к первоначальной настройке

Запуск сервиса осуществляется командой systemctl start tiamat-manager.service
Перезапуск сервиса осуществляется командой systemctl restart tiamat-manager.service
Проверка статуса сервиса осуществляется командой systemctl status tiamat-manager.service

Конфигурация Tiamat Hooks

Конфигурация пакета хранится в файле
Debian: /etc/tiamat/appsettings.json

Все нижеуказанные настройки выполняются в нем.

Первоначальная конфигурация

После установки в первую очередь необходимо настроить соединение с consul и gitlab. Для это нужно в файле настроек выполнить следующее:

1. В разделе Consul заполнить поля:
    1. **Host** - адрес до сервера consul
    2. **RootKey** - корневой ключ конфигурации Tiamat, который был сформирован после установки Consul
    3. **Token** - токен доступа для Consul, настроенный на чтение конфигурации из корневого ключа.
2. В разделе Gitlab заполнить поля:
    1. **Host** - адрес до сервера Gitlab
    2. **Token** - api токен для доступа к Gitlab с правами api, read_user, read_repository.

После этого сервис можно перезапускать и пользоваться.
Сервис будет доступен по адресу

Конфигурация Tiamat Manager

Конфигурация сервиса хранится в файле
Debian: /etc/tiamat/appsettings.json

Все нижеуказанные настройки выполняются в нем.

Первоначальная конфигурация

После установки в первую очередь необходимо настроить соединение с базой данных и jira. Для это нужно в файле настроек выполнить следующее:

1. В разделе ConnectionStrings в подразделе Default указать полную строку подключения в виде “*server=localhost;UserId=postgres;Password=пароль указанный при установке Postgresql;database=tiamat;”*
2. В разделе Consul заполнить поля:
    1. **Host** - адрес до сервера consul
    2. **RootKey** - корневой ключ конфигурации Tiamat, который был сформирован после установки Consul
    3. **Token** - токен доступа для Consul. Token должен предоставлять доступ на редактирование для корневого ключа. Например,

```
key_prefix "services/tiamat" {
  policy = "write"
}
```

3. В разделе Gitlab заполнить поля:
    1. **Host** - адрес до сервера Gitlab
    2. **Token** - api токен для доступа к Gitlab с правами api, read_user, read_repository.

После этого сервис можно перезапускать и пользоваться.
Сервис будет доступен по адресу http://<ip адрес сервера>:8050/

Правила

Правила это состояния коммит-хуков для определенного репозитория. Наименование правила соответствует наименование указанного репозитория. Позволяют поддерживать целостность репозитория и избегать проблем, выполняя проверки коммитов перед пушем. Список правил обновляется автоматически (ежечасно) на основе списка репозиториев гитлаба.

Правила

Элементы:

1. Панель навигации.
2. Список репозиториев.
3. Карточка хука. Содержит:
    1. Кнопку-переключатель;
    2. Название хука;
    3. Автора или ответственного за хук;
    4. Ссылку на соответствующую страницу хука в Confluence;
    5. Описание;
    6. Кнопку настройки хука.
4. Управляющие кнопки:
    1. Сохранить. Фиксирует изменения в текущем репозитории;
    2. Тест. Позволяет проверить заданную цепочку коммитов хуками. Дебаг-функция.
    3. Репорты. Позволяет смотреть результаты теста. Дебаг-функция.
5. Вход.

Как найти/включить хук?

1. Перейдите к репозиторию проекта, выбрав его в списке слева.
2. Найдите хук в общем списке.
3. Переключите кнопку на карточке хука.
4. Сохраните кнопкой вверху страницы.

Как увидеть список всех хуков на проекте?

1. Перейдите к репозиторию проекта, выбрав его в списке слева.
2. Найти все хуки во "включенном" состоянии.

Эталоны

Эталонные состояния правил необходимы для поддержания состояния правила в необходимом проекту виде. В эталон включаются необходимые коммит-хуки, которые обязательно должны быть включены для выбранного репозитория. Задаются сотрудниками с определенными правами.
Эталоны являются основой для валидации о расхождении текущего состояния правила с эталонным в отрицательную сторону, проще говоря, если какой-то либо коммит-хук указанный в эталоне отключен, то это является ошибкой. По результатам валидации отправляется нотификация в канал #alerts

Как добавить/отредактировать эталон?

1. Перейдите к репозиторию проекта, выбрав его в списке слева.
2. Нажмите кнопку эталон в заголовке списка хуков
3. Выберите необходимые хуки, в открывшемся диалоговом окне
4. Нажмите кнопку "Сохранить эталон" внизу окна

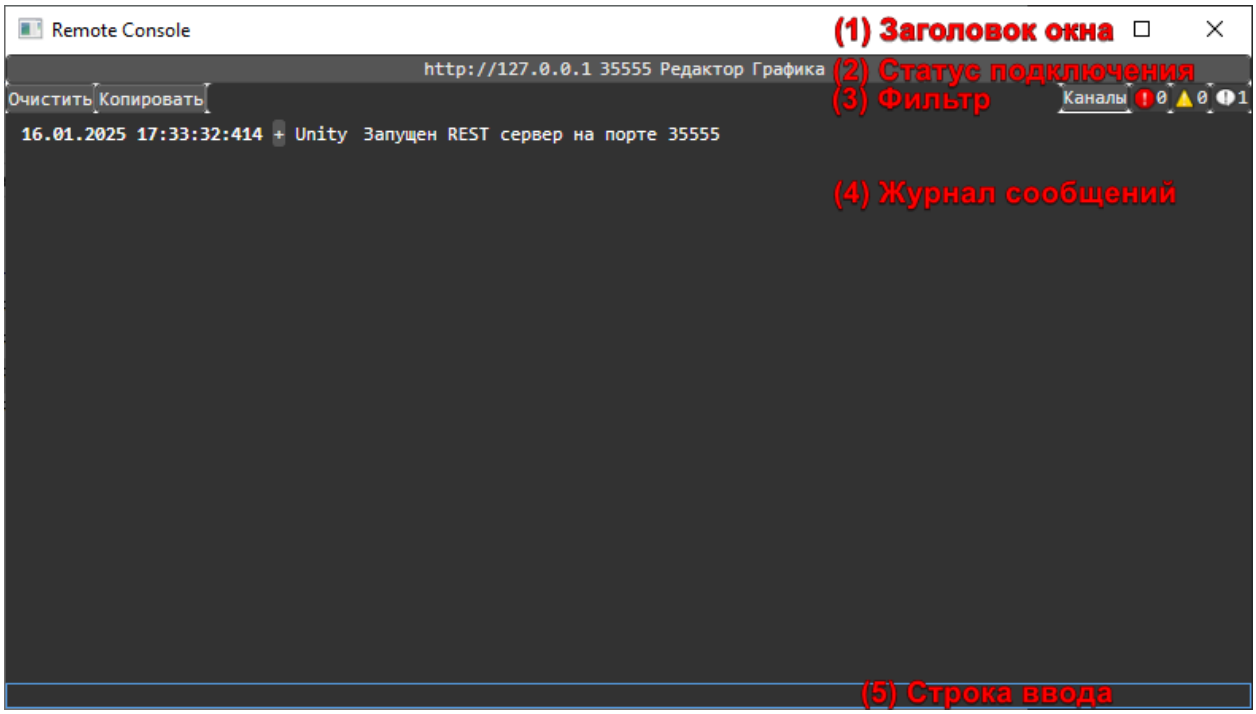
Remote Console

Описание: программа удаленного управления игровыми приложениями

Система состоит из двух частей:

- Интеграция в движок (в данном случае – Unity)
- Внешний исполняемый файл – `RemoteConsole.exe`

Внешний вид окна Remote Console



Запуск

1. Убедиться, что на компьютере установлен редактор Unity версии `2022.3.7f1`.
2. Открыть в Unity проект `remoteconsole-example-project`.
3. Убедиться, что компиляция прошла успешно: отсутствуют сообщения об ошибке в консоли.
4. Убедиться, что сервер консоли поднялся: в консоли есть сообщение:
   `Запущен REST сервер на порте <...>`
5. Запустить `RemoteConsole.exe`.
6. Убедиться, что консоль подключилась к редактору:
   a. В журнале сообщений (4) Remote Console появилось сообщение из Unity:
      `Запущен REST сервер на порте <...>`
   b. В статусе подключения (2) Remote Console указан тот порт, на котором поднят сервер (обычно 35555).

Работа

1. Выполнить тестовую команду:
    a. В строке ввода (5) Remote Console ввести текст `test_command`.
    b. Подтвердить кнопкой `Enter`.
2. Убедиться, что команда выполнена:
    a. В журнале сообщений (4) Remote Console добавились строки вида:
    `“Принята команда: test_command аргументы:”`
    и
    `“Результат выполнения: Режим игры РЕДАКТОР”`
3. Переключить редактор в режим Play.
4. Выполнить тестовую команду:
    a. В строке ввода (5) Remote Console ввести текст `test_command`.
    b. Подтвердить кнопкой `Enter`.
5. Убедиться, что команда выполнена:
    a. В журнале сообщений (4) Remote Console добавились строки вида:`“Принята команда: test_command аргументы:”`
    и
    `“Результат выполнения: Режим игры ИГРА”`
6. Полный список всех известных команд и переменных можно увидеть, выполнив команду `help`.