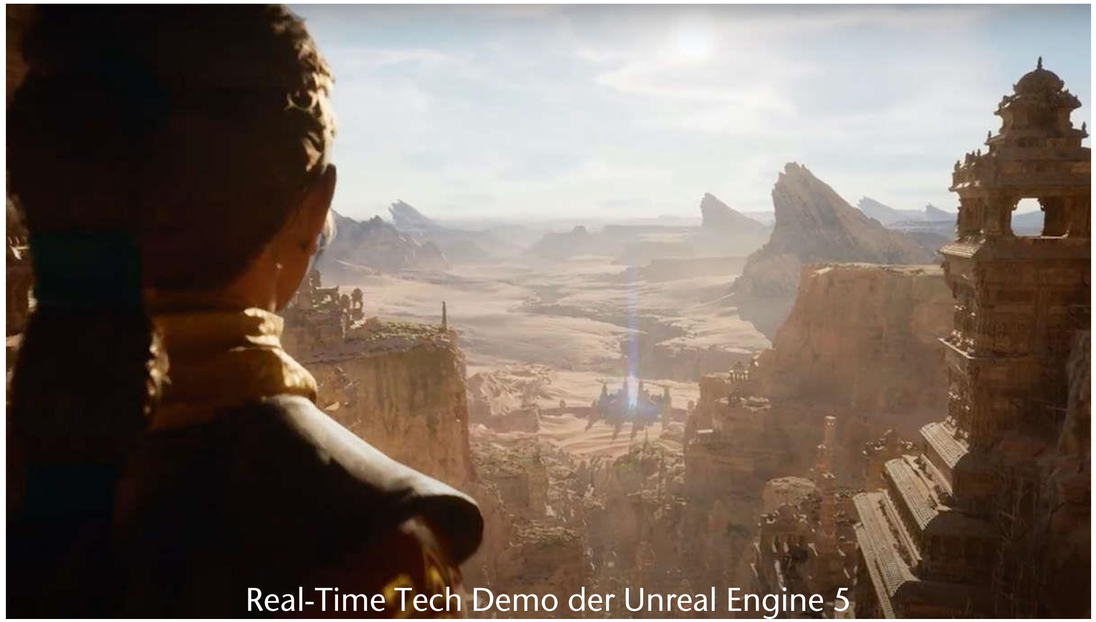
Introduction to the Unreal Engine 5



Real-Time Tech Demo der Unreal Engine 5

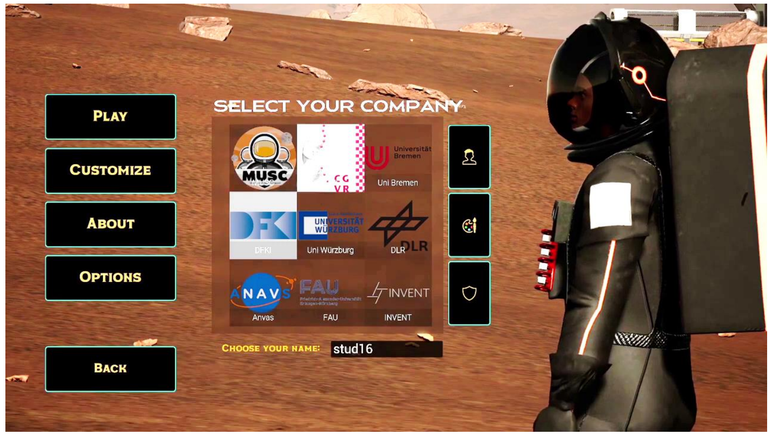
A. Mühlenbrock, CGVR, Universität Bremen

Overview

- What is the Unreal Engine 5?
  - A Game Engine / Graphics Engine
    - Primarily for 3D applications
      - 2D games are also possible (see Paper 2D).
      - **Competitors:** Unity, Godot (free).
    - **Licence:** Free to use up to $100,000 in revenue per year, 5% of revenue thereafter

Famous games

Applications from CGVR



Dental Sim

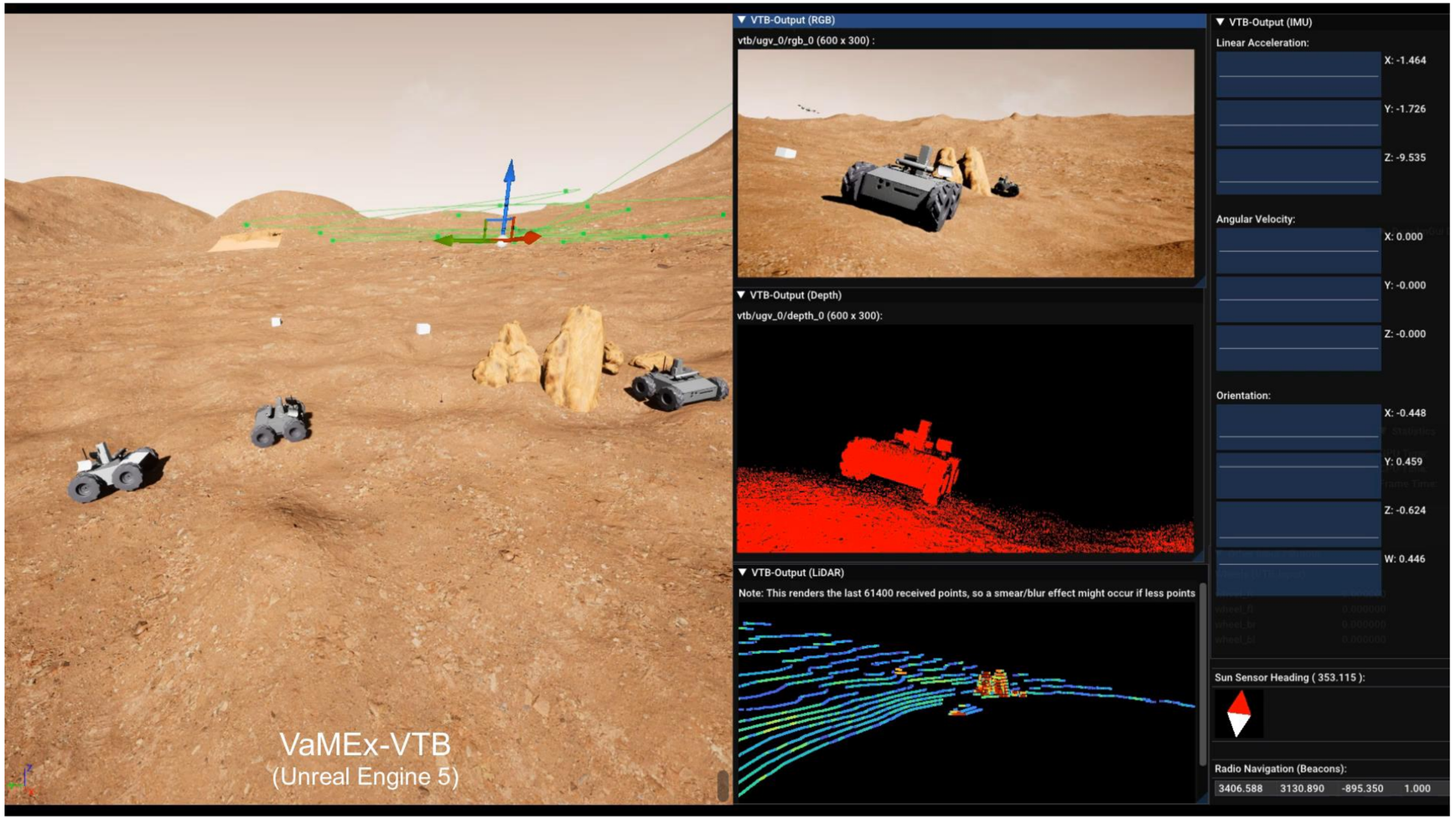
Coral Reef VR

VaMEx VTB / MUSC

Applications from CGVR

Dental Sim

Applications from CGVR



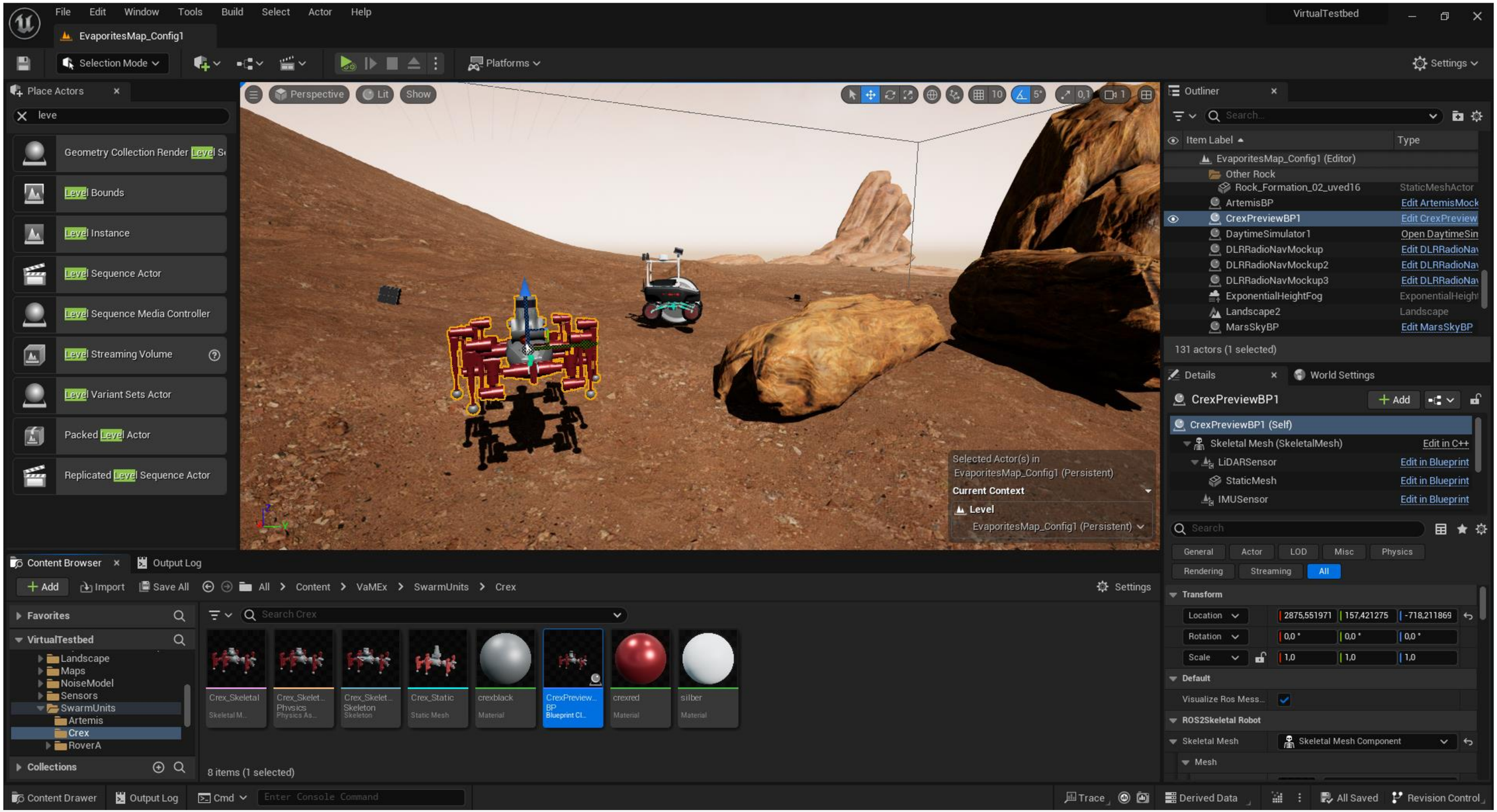
VaMEx-VTB

Engine vs. Editor

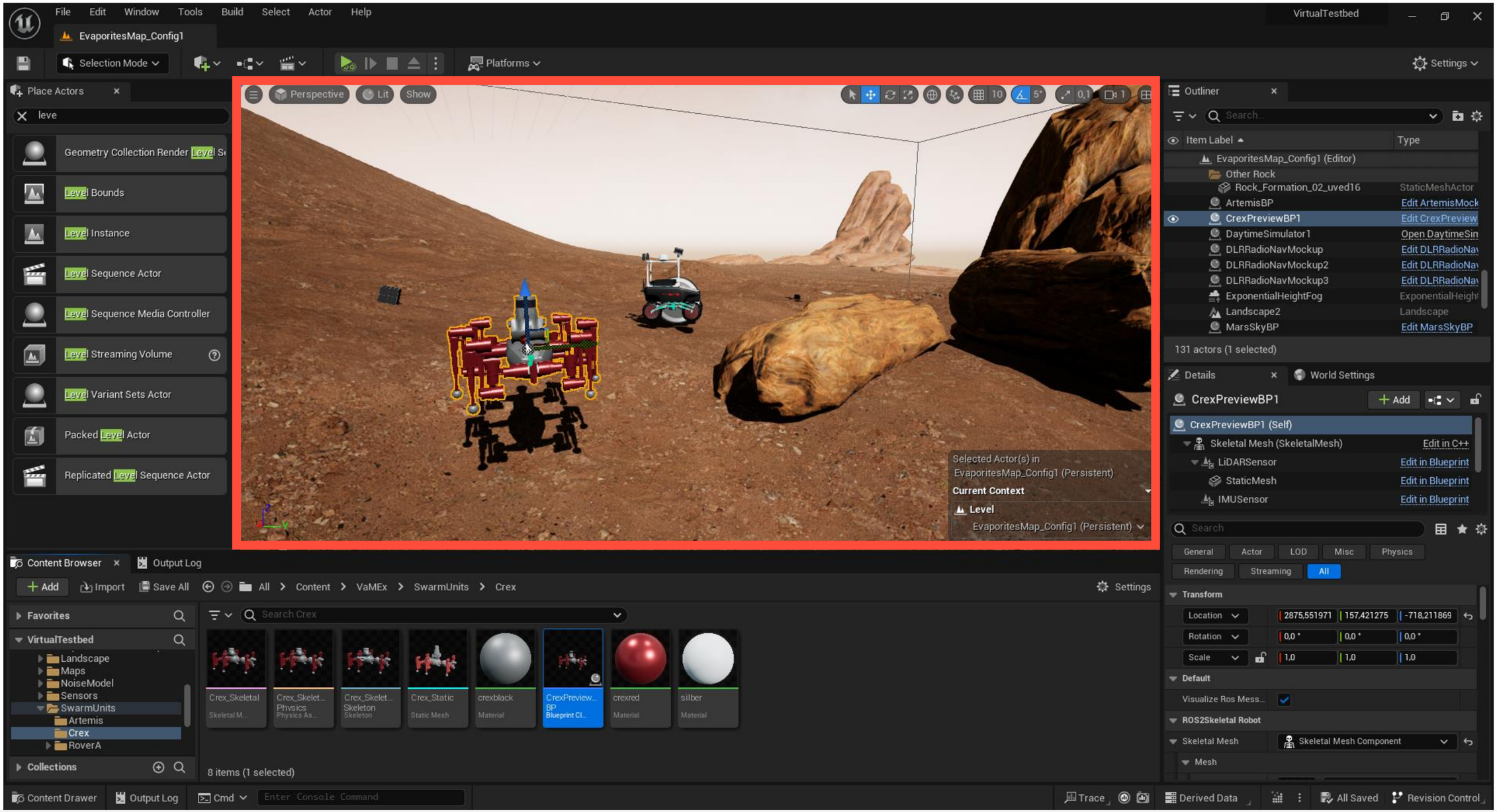
- Unreal Engine
  - Contains classes for rendering, physics simulation, audio playback, etc.
  - Is shipped with the developed game
- Unreal Editor
  - Powerful tool for developing games with Unreal Engine
    - Design levels, materials, blueprints and animate.
  - Is **not** shipped with the developed game

Unreal Editor

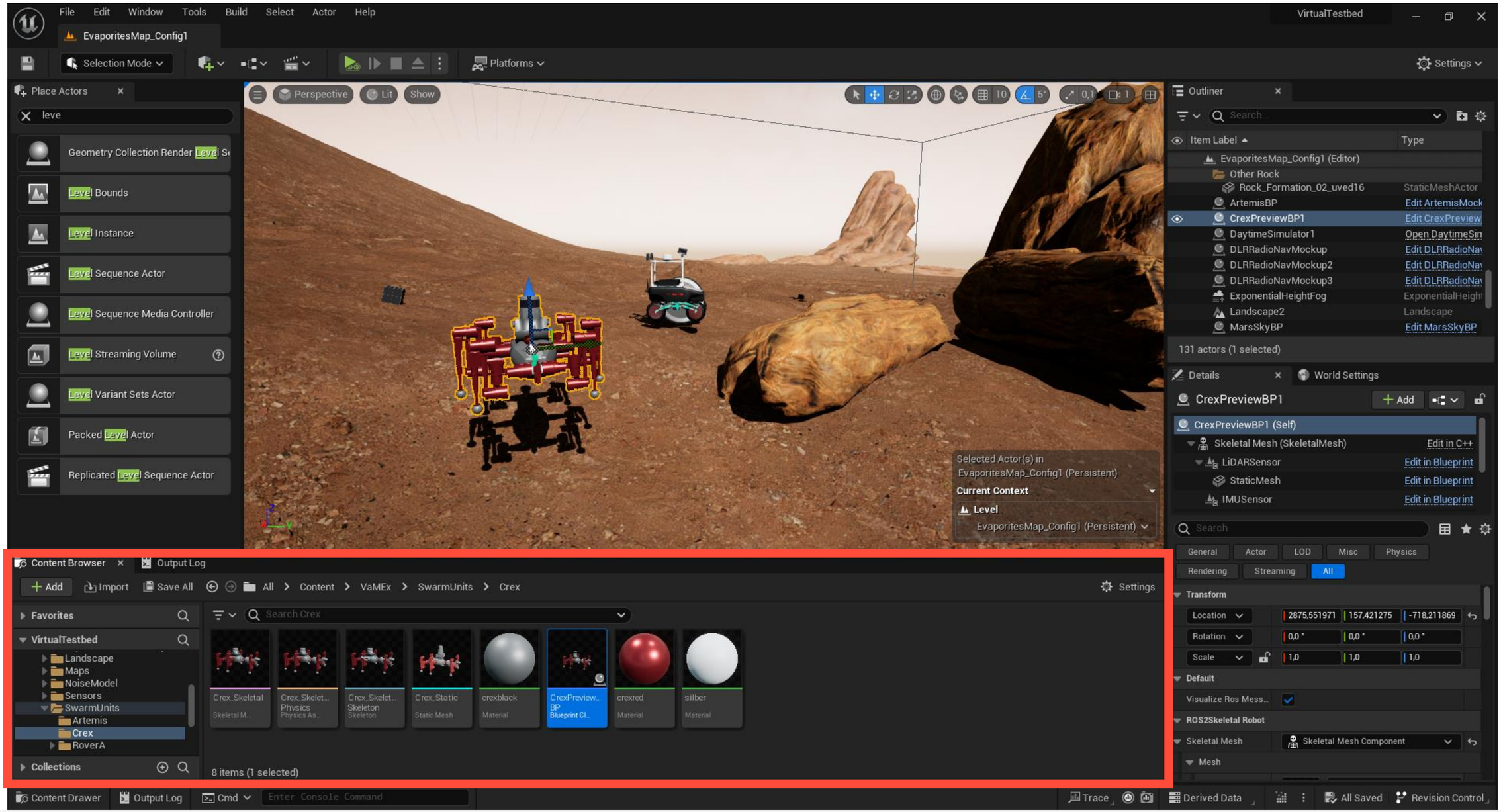
Unreal Editor



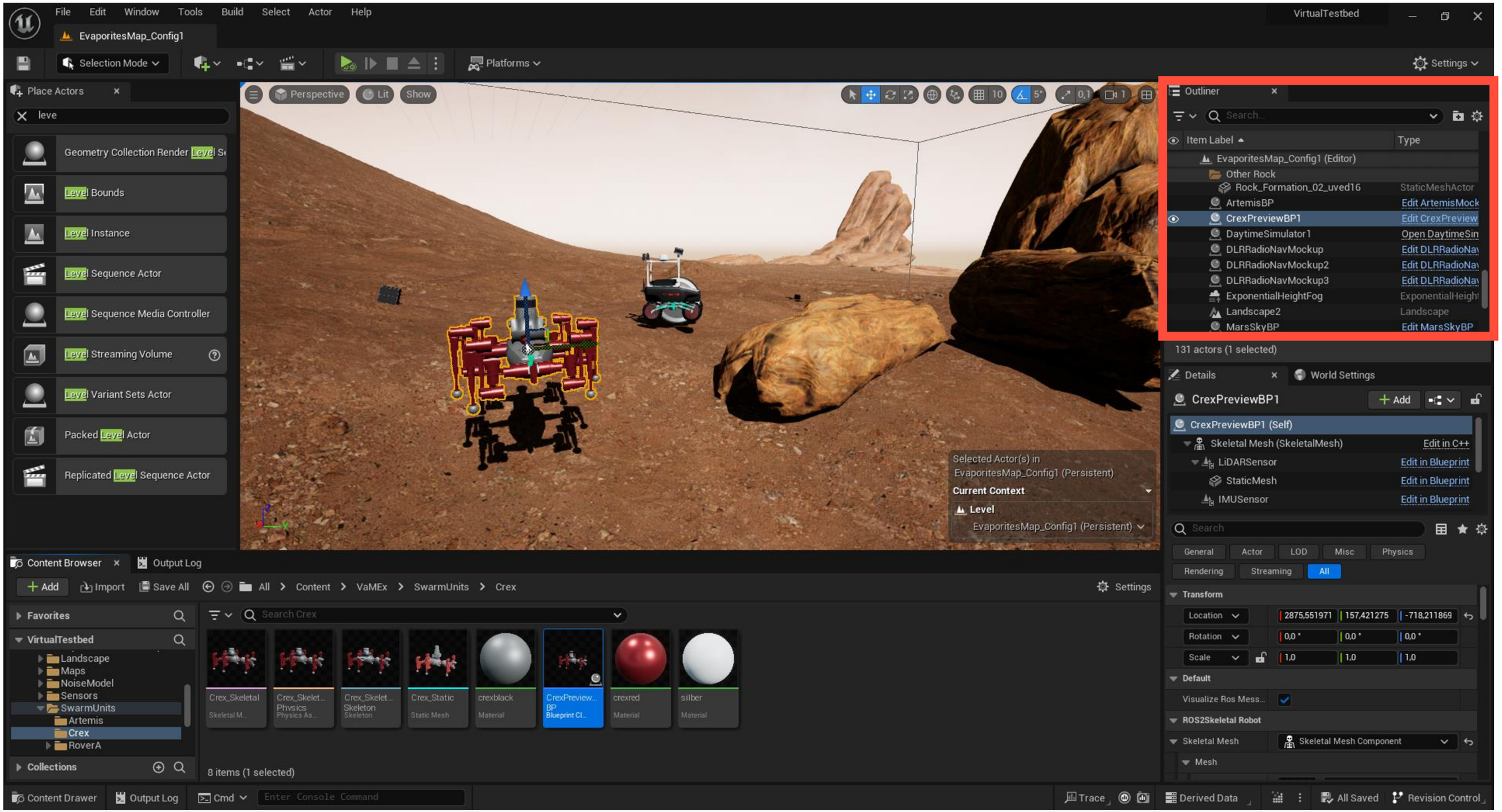
Unreal Editor



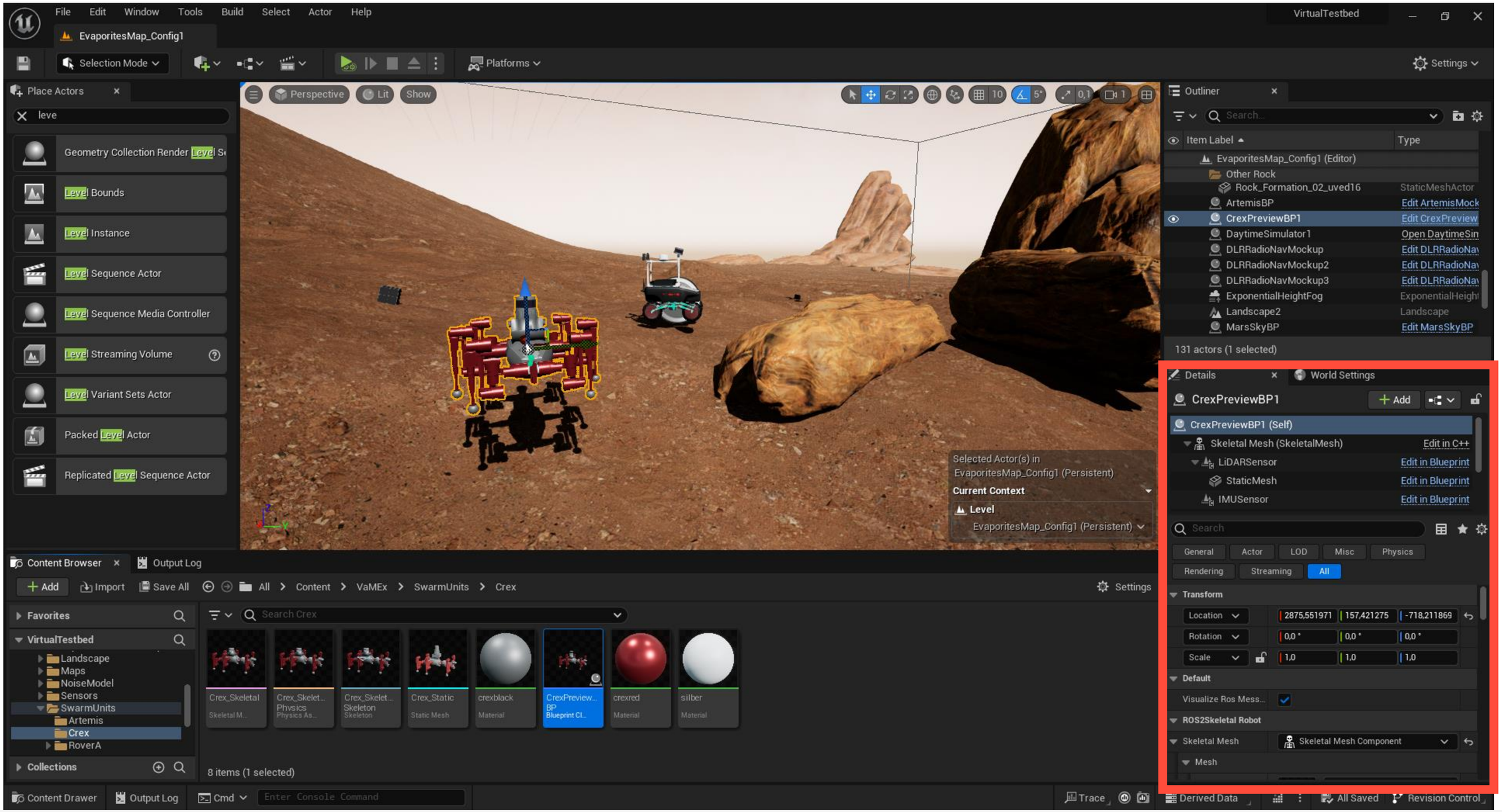
Unreal Editor



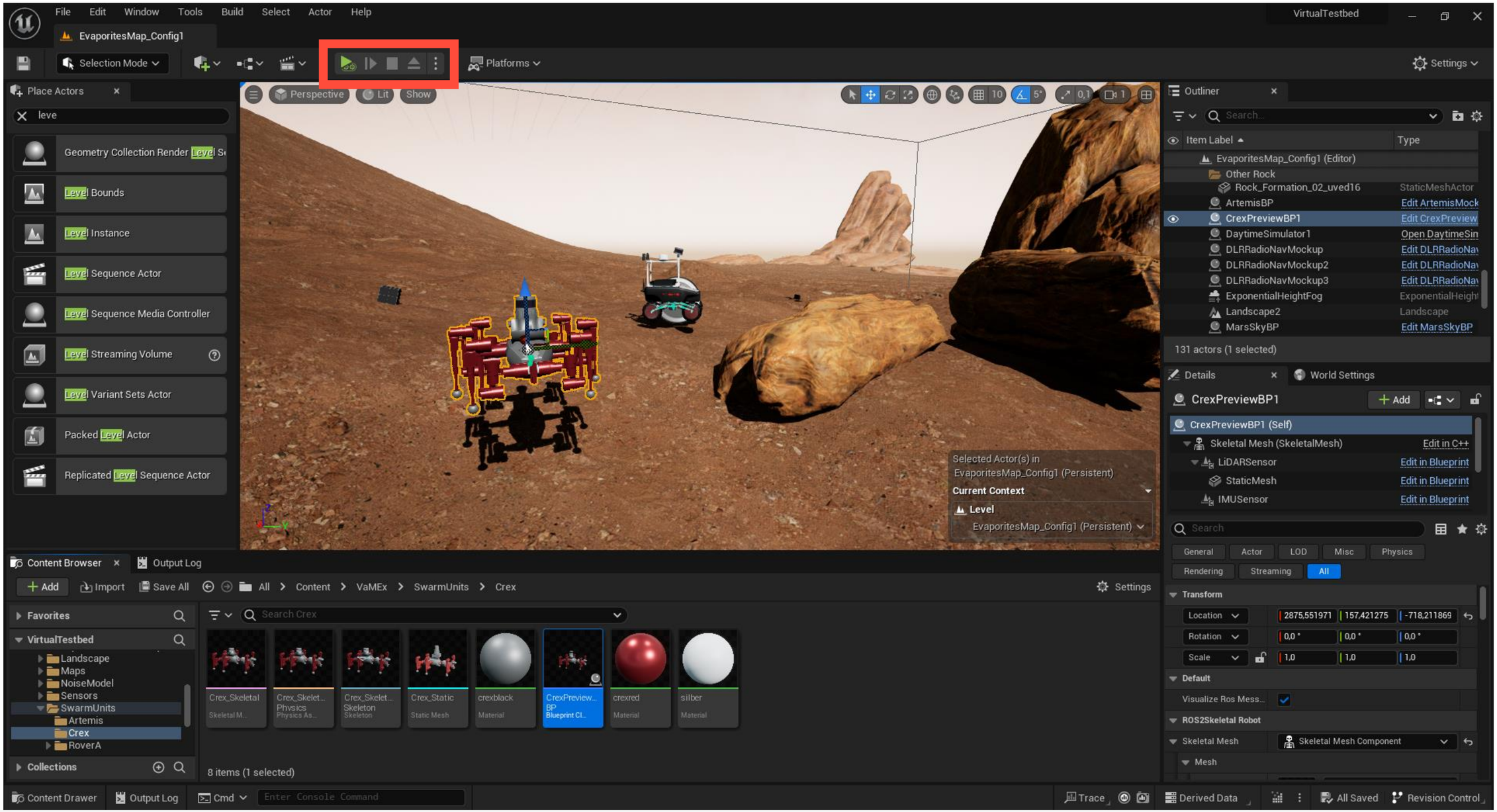
Unreal Editor



Unreal Editor



Unreal Editor



Programming using the Unreal Engine

Engine: Programming

- How to program in Unreal Engine 5?
  - **C++ (Code)**
    - Using Visual Studio (Windows), …
  - **Blueprints (Visual Programming)**
    - Directly in Unreal Editor

➢ Both can be combined!

➢ Blueprint classes can even inherit from C++ classes

Engine: Programming

- **C++ Code**

  + For complex code, often more readable
  + Simpler / faster implementation of math equations
  + Faster execution
  - Slower development

- **Blueprints**

  + Faster development
  + Simple adding of additional functionality to C++ classes
  - Tend to get messy
  - Slower execution
  - Not all C++ functions available

Hands On

Throwing you in at the deep end

Creating a custom Actor

(Blueprint version)

“Ein Wurf ins kalte Wasser”

Three essential base classes

(both Blueprint and C++)

Engine: Essential base classes

- Three essential base classes during development:
  1. Level
  2. Actor
  3. Component

Engine: Essential base classes

1. Level

- A „game world“
- Can be considered as a collection of objects.
  - An object that can be placed into the level is called **Actor**

➢ At any time, a level must be loaded

- Even if it's an almost empty level for the start menu of a game.

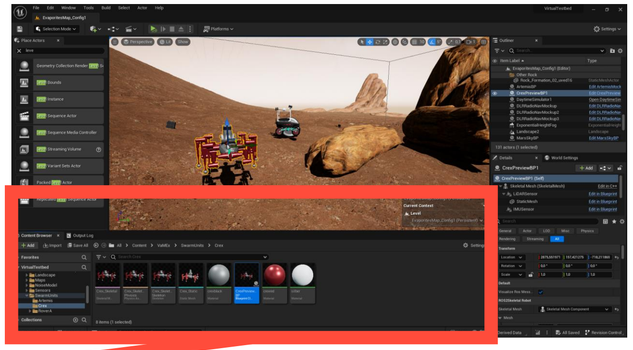
Engine: Essential base classes

2. **Actor** (1/2)

- Objects that can be added to a level
  - Typical examples:
    - A game character
    - A wall, a floor, …
    - Program code (Services, Managers, …)

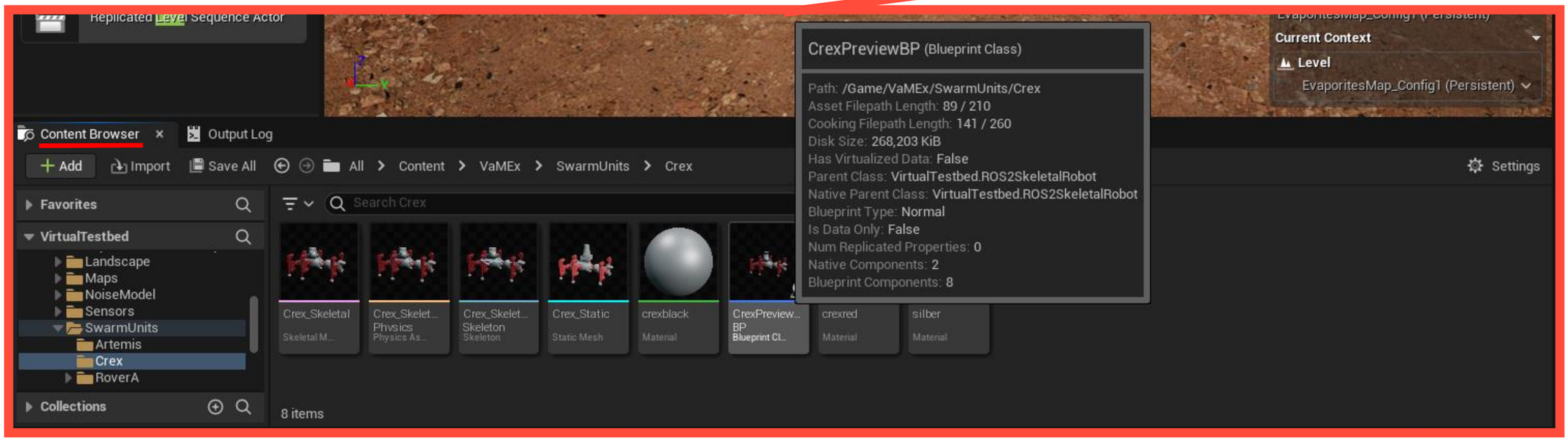
➢ Object-oriented: An actor is defined as a class, objects of this class can then be placed into levels.

Engine: Essential base classes

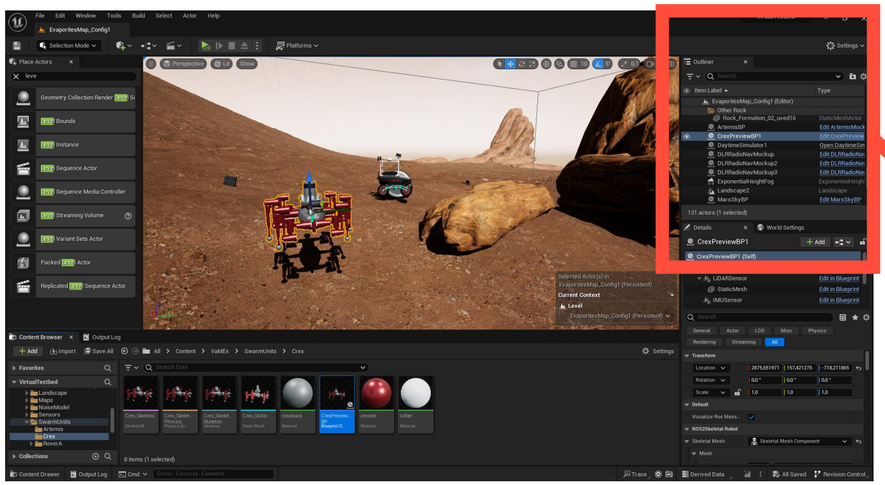


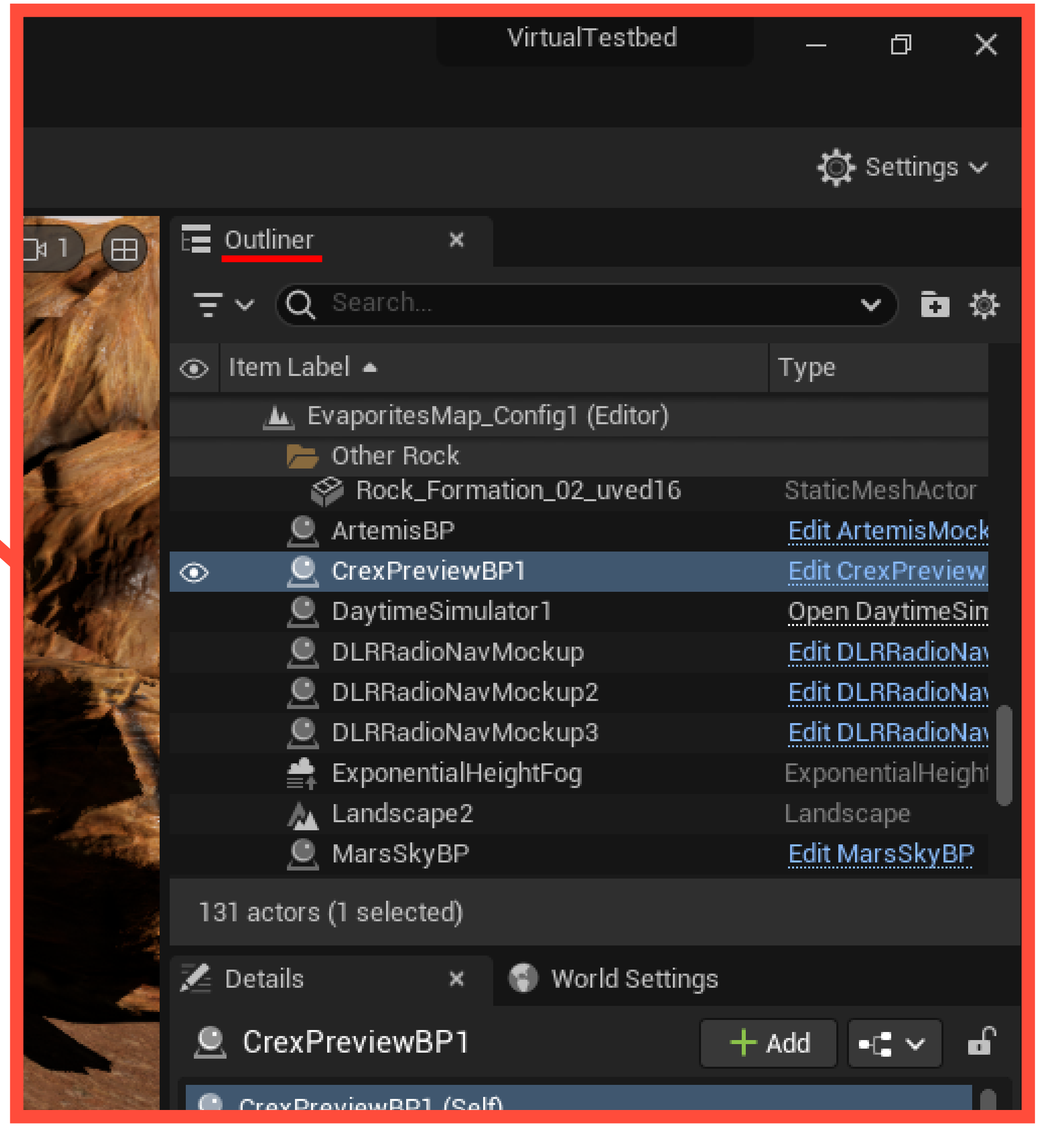
In the **Content Browser** you can find and create

Actor **classes** (among many other things)



Engine: Essential base classes





Outliner shows all Actor **objects** in the current loaded level

Engine: Essential base classes

2. **Actor** (2/2)

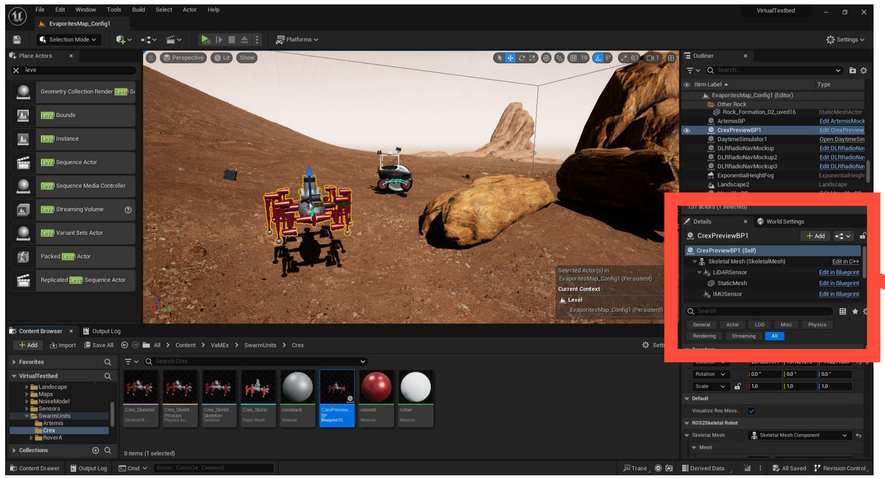
- Actors have functions that can be overwritten:
  - **Tick():** Is called every frame
    - Can be used for simple animation, processing control input, etc.
  - **BeginPlay():** Called when game started (or an actors is dynamically placed into the level)
  - **EndPlay():** Called when game ends (or an actor gets destroyed)
  - …

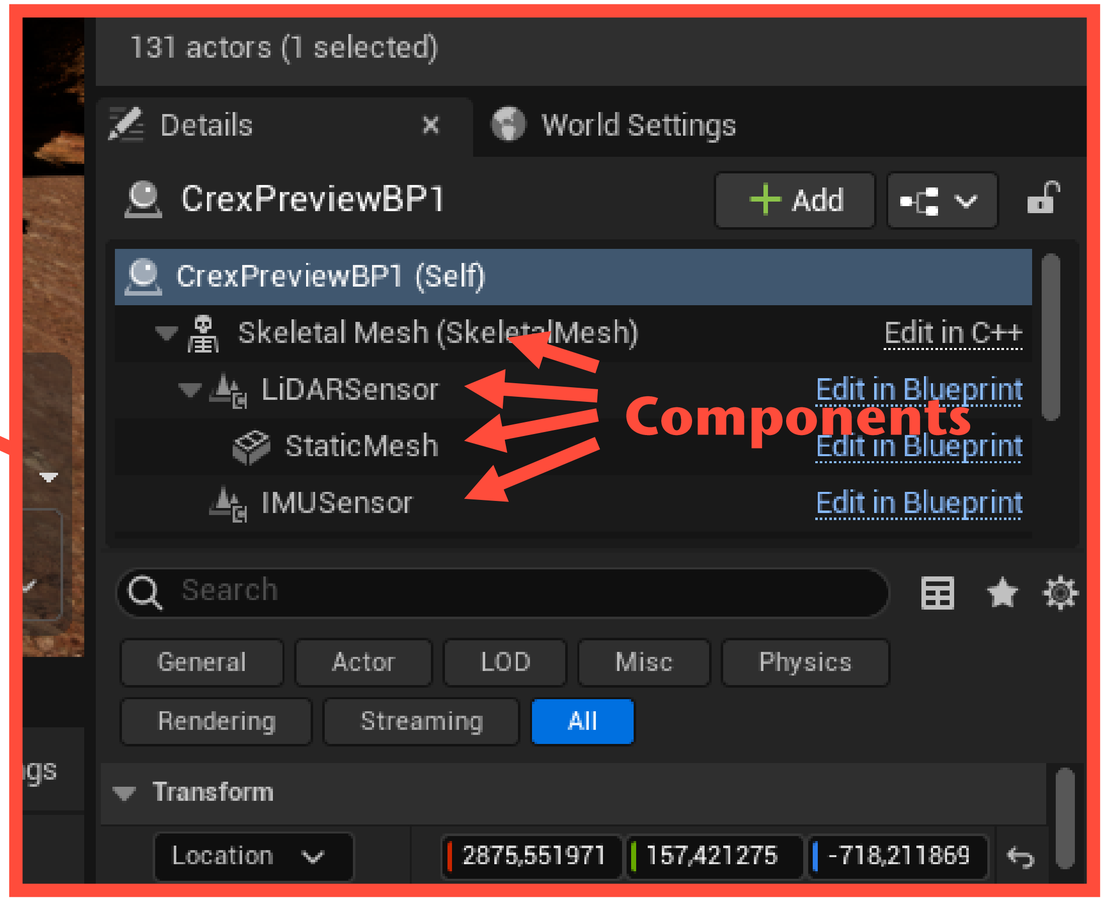
Engine: Essential base classes

3. **Components** (1/2)

- Objects that can be added to an Actor.
- A component adds functionality to that Actor.
- Examples:
  - A mesh
  - A light
  - Reused program code
  - A scene component

Engine: Essential base classes





Detail panel shows all component **objects** of an actor object

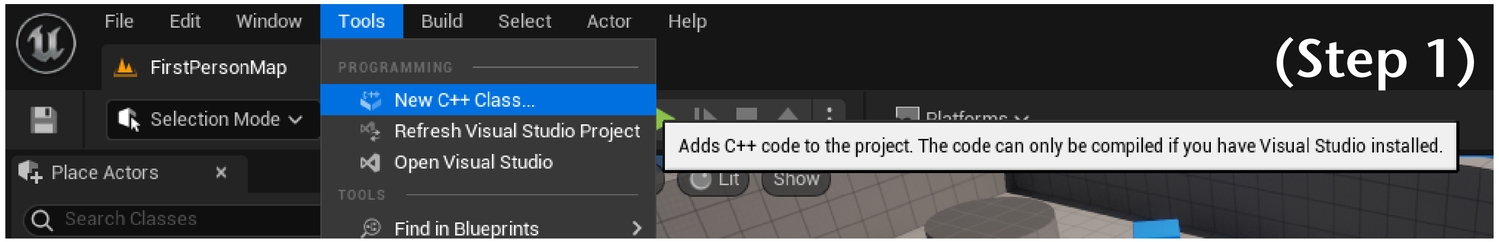
Engine: Essential base classes

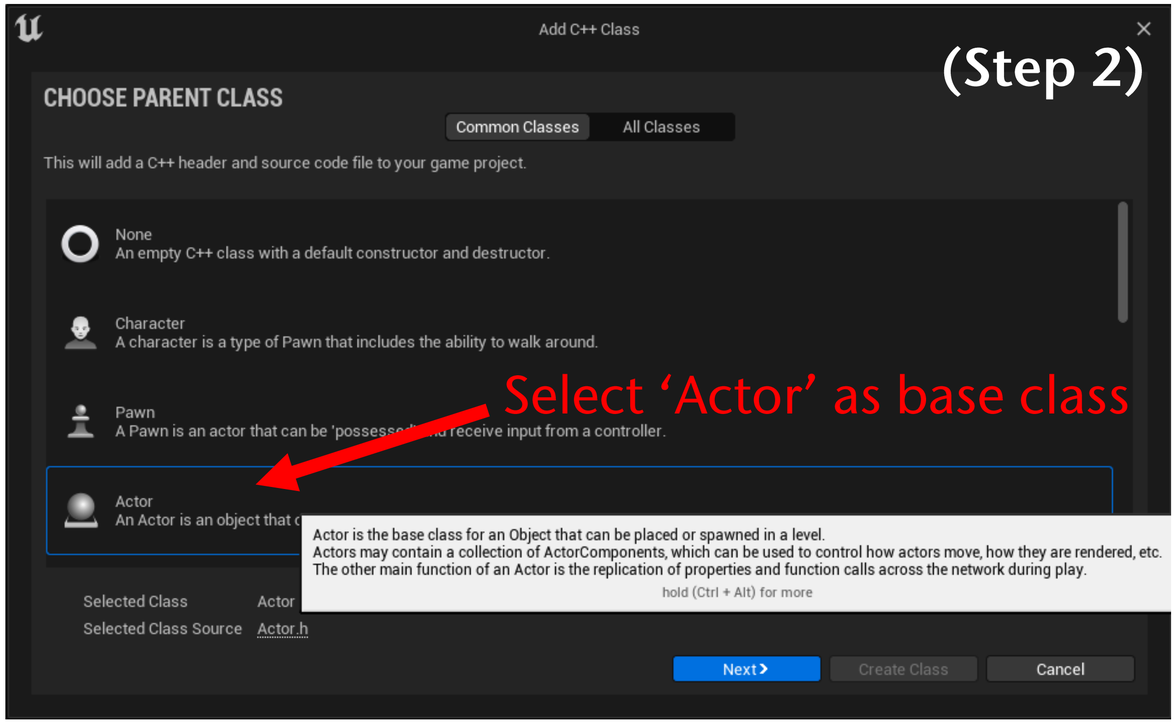
3. Components (2/2)

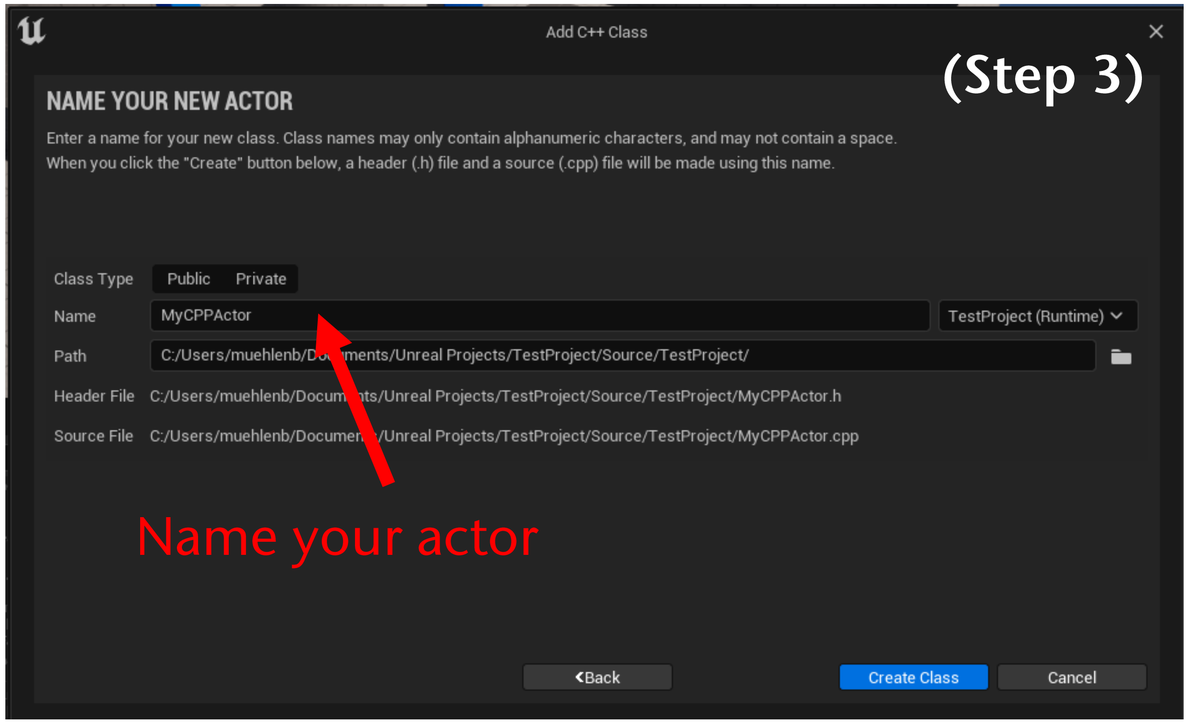
- Components can be hierarchically arranged in an Actor.
  - Works like a scene graph
- An actor should have a root component.
  - The root component stores the **position, rotation** and **scale** of the whole actor!
  - A new Blueprint Actor has a “DefaultSceneRoot” which is a Scene Component.
    - ➢ In C++, there is no default RootComponent.
    - ➢ If you don’t create a Scene Component in C++ by your own, your actor has no location!

Creating a new C++ Actor

Creating a new C++ Actor







Structure of a C++ Actor

Creating a C++ Class

MyActor.h

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "MyActor.generated.h"

UCLASS()
class TESTPROJECT_API AMyActor : public AActor
{
    GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    AMyActor();

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

};
```

Naming convention: The class name of an actor starts always with a ‘A’

Constructor

BeginPlay function

Tick funktion

Creating a C++ Class

MyActor.h

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "MyActor.generated.h"

UCLASS()
class TESTPROJECT_API AMyActor : public AActor
{
    GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    AMyActor();

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

};
```

Macros: Makes the class visible for the Unreal Editor (not really important to understand)

Creating a C++ Class

MyActor.h

```
class TESTPROJECT_API AMyActor : public AActor
{
    GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    AMyActor();

    float progress;

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

};
```

Creating a C++ Class

MyActor.h

```
class TESTPROJECT_API AMyActor : public AActor
{
    GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    AMyActor();

    float progress;

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

};
```


Defining a property.

Creating a C++ Class

MyActor.h

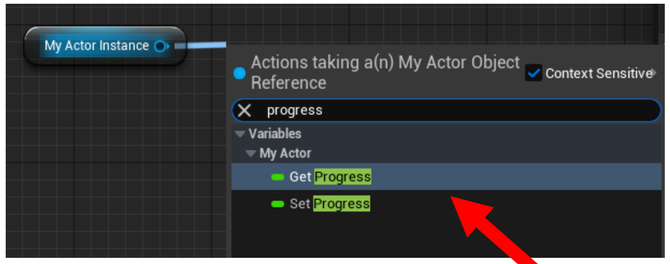
```
class TESTPROJECT_API AMyActor : public AActor
{
    GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    AMyActor();

    float progress;

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

};
```

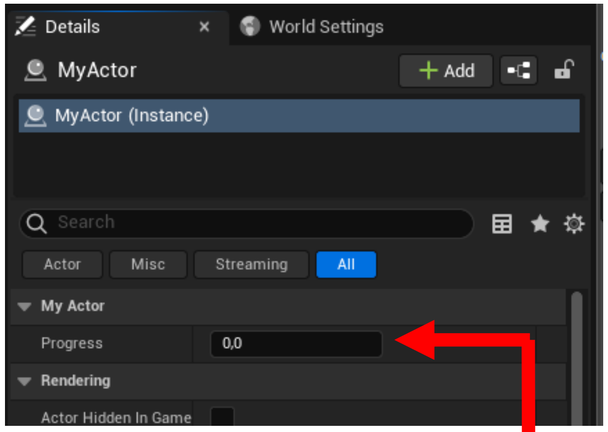

But this is not visible in the Unreal Editor or in blueprints, too ☹

Creating a C++ Class

MyActor.h

```
class TESTPROJECT_API AMyActor : public AActor
{
    GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    AMyActor();

    UPROPERTY(BlueprintReadWrite, EditAnywhere)
    float progress;

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

};
```





But we can make it visible ☺

BlueprintReadWrite: Allows access to it in blueprints

EditAnywhere: Allows configuration in the details panel.

Hands On

Creating a custom Actor

(C++ version)

More about **Actors** and **Components**

Actor Class

- **AActor** (C++ class):
  - Base class of all actors (no matter if Blueprint or C++).
    - Every further Actor class inherits from it
  - Many useful Actor classes exist, e.g.:
    - **AStaticMeshActor**: Allows to place a static mesh into the level (e.g. a wall, a floor)
    - **APawn**: Actor that can be "possessed“ (e.g. a vehicle)
      - I.e. you can see from its perspective and control it.
    - **ACharacter**: An **APawn** that also contains a **skeletal mesh** and **movement logic**.

Actor Class

- **Note:** Functionality is often implemented in a component class, not in the actor class!

- Example:
  - AStaticMeshActor:
    - Contains a UStaticMeshComponent in which you can select the StaticMesh.

Component Class

- Often used component classes:
  - UStaticMeshComponent
  - USkeletalMeshComponent
  - ULightComponent
    - UPointLightComponent, UDirectionalLightComponent, ...
  - UCameraComponent
  - ...

Component Class

- There are corresponding actor classes (with one component each):
  - AStaticMeshActor
  - ASkeletalMeshActor
  - ALight
    - APointLight, ADirectionalLight, …
  - ACameraActor
  - …

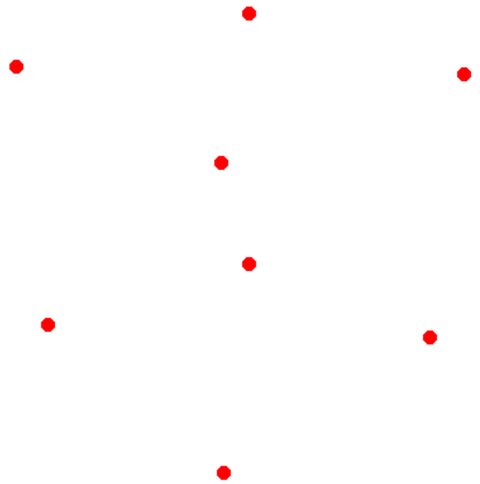
Insights into other relevant topics

In a 3D game, we need 3D objects...

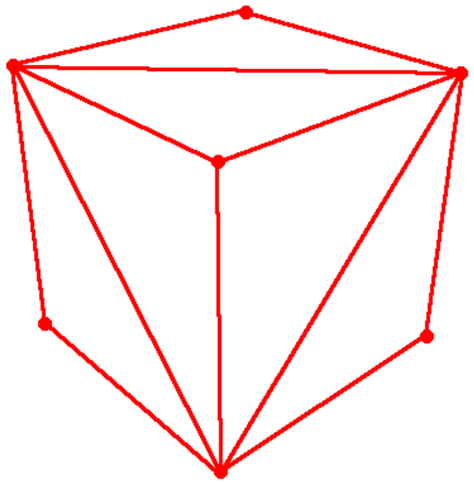
Recap: Meshes

Recap: Mesh

- Defined by:
  - Vertices (“Eckpunkte”)

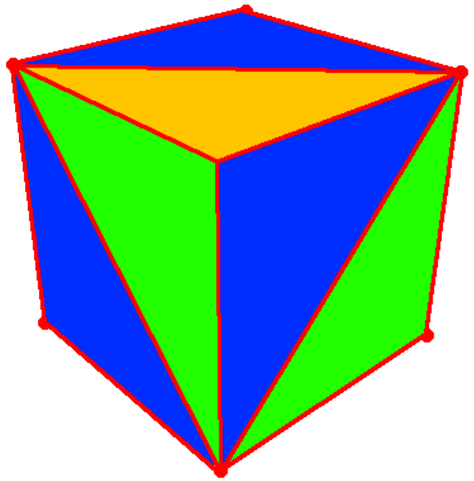


Recap: Mesh



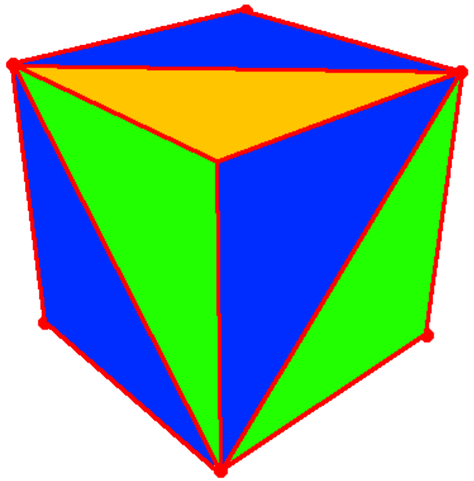
- Defined by:
  - Vertices (“Eckpunkte”)
  - Edges (“Kanten”)

Recap: Mesh



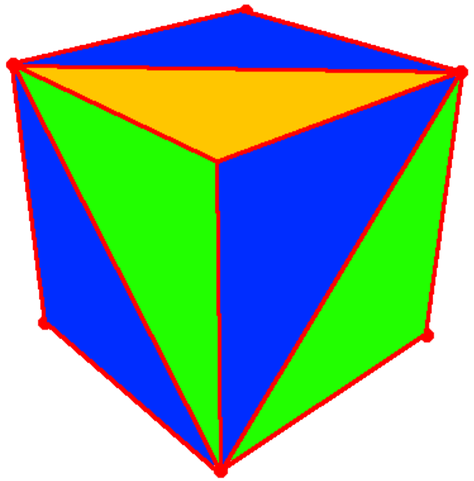
- Defined by:
  - Vertices (“Eckpunkte”)
  - Edges (“Kanten”)
  - Faces (“Flächen”)

Recap: Mesh



- Defined by:
  - Vertices (“Eckpunkte”)
  - Edges (“Kanten”)
  - Faces (“Flächen”)
    - Often triangles

Recap: Mesh

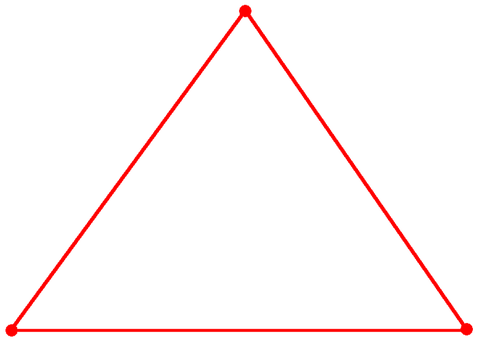


- Defined by:
  - Vertices (“Eckpunkte”)
  - Edges (“Kanten”)
  - Faces (“Flächen”),
    - Often triangles

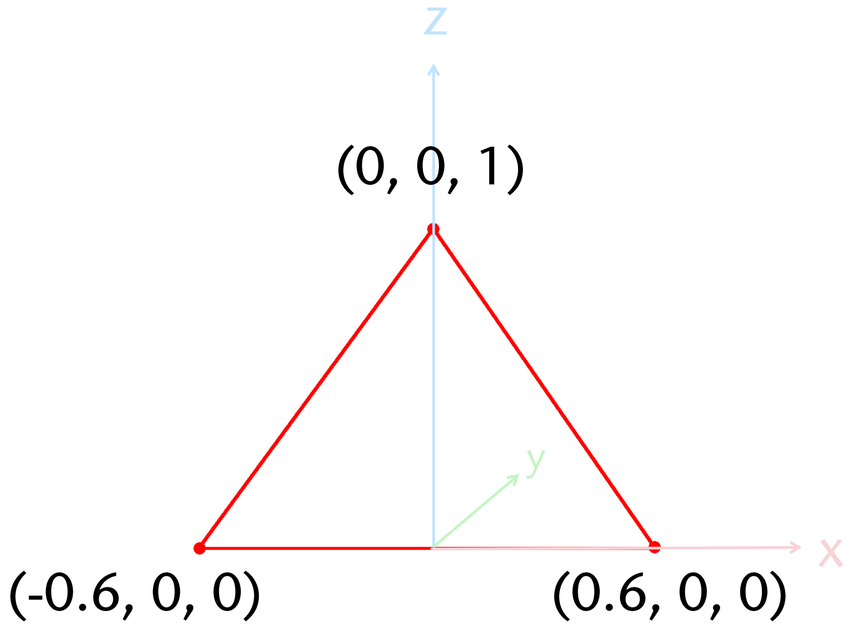
➢ Such a mesh is usually used for 3D models in the Unreal Engine.

Recap: Vertex / Vertices

- Vertices store attributes (“Vertex attributes”)

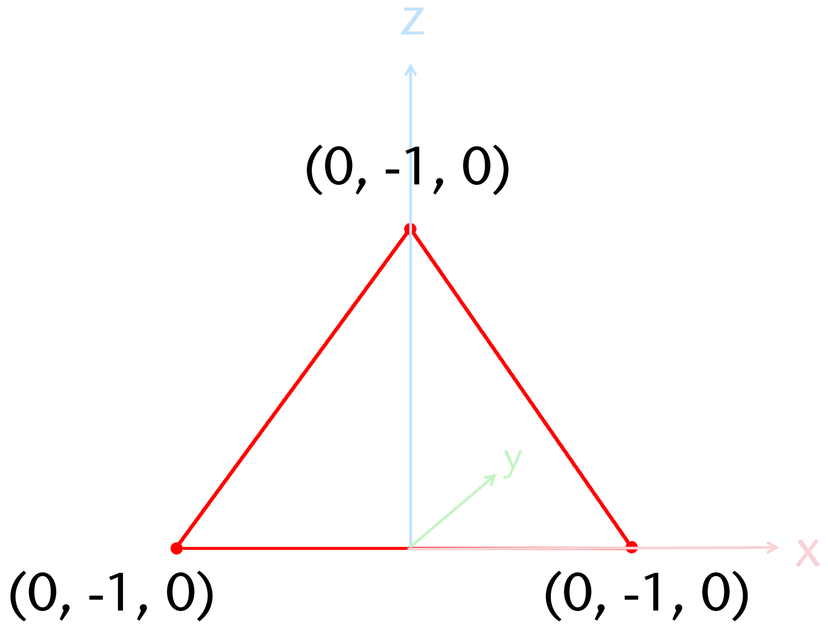


Recap: Vertex / Vertices



- Vertices store attributes (“Vertex attributes”):
  - Position

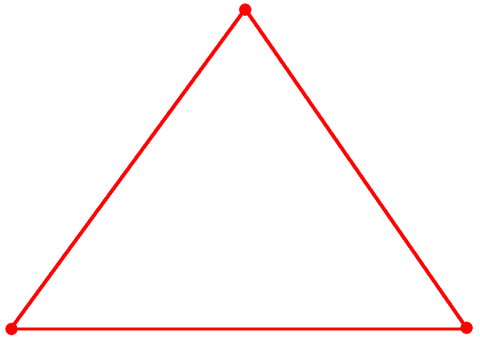
Recap: Vertex / Vertices



- Vertices store attributes (“Vertex attributes”):
  - Position
  - Normal
    - Here, it points towards the screen (negative y axis)

Recap: Vertex / Vertices

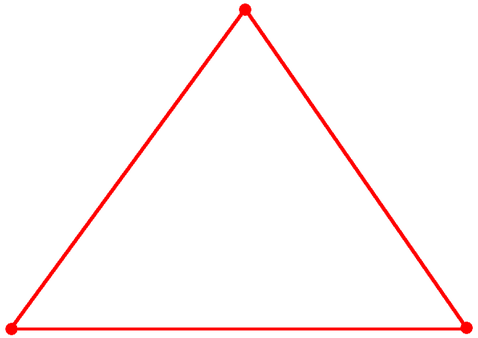
3D mesh



- Vertices store attributes (“Vertex attributes”):
  - Position
  - Normal
    - Here, it points towards the screen (negative y axis)
  - UV coordinates
    - Coordinates in the 2D texture

Recap: Vertex / Vertices

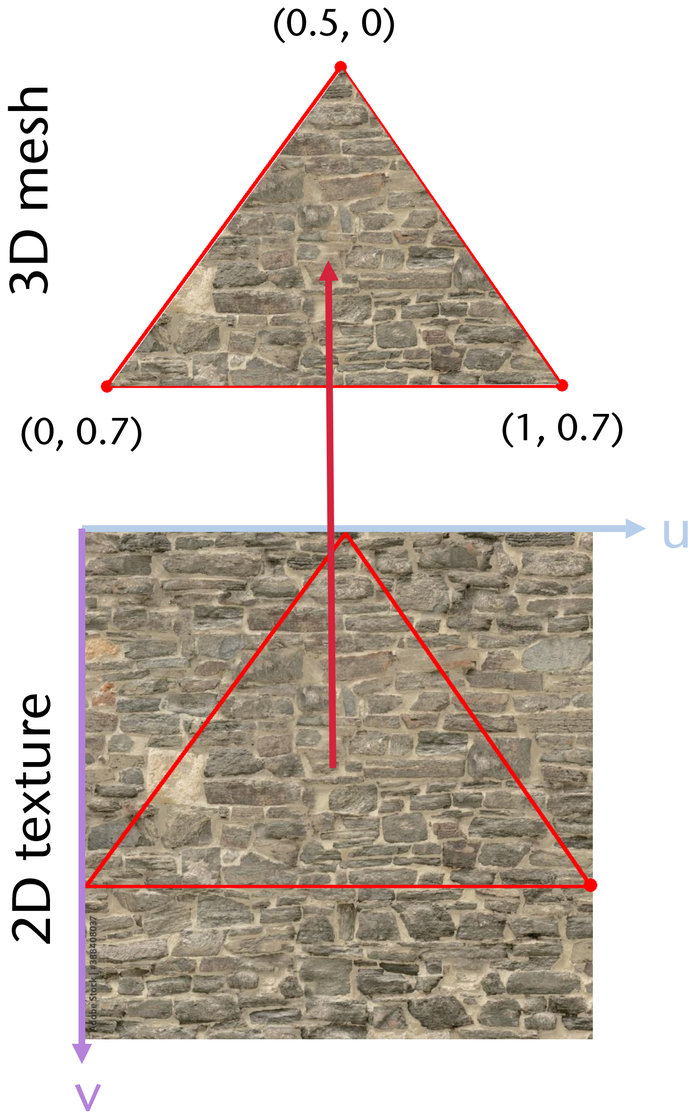
3D mesh



2D texture

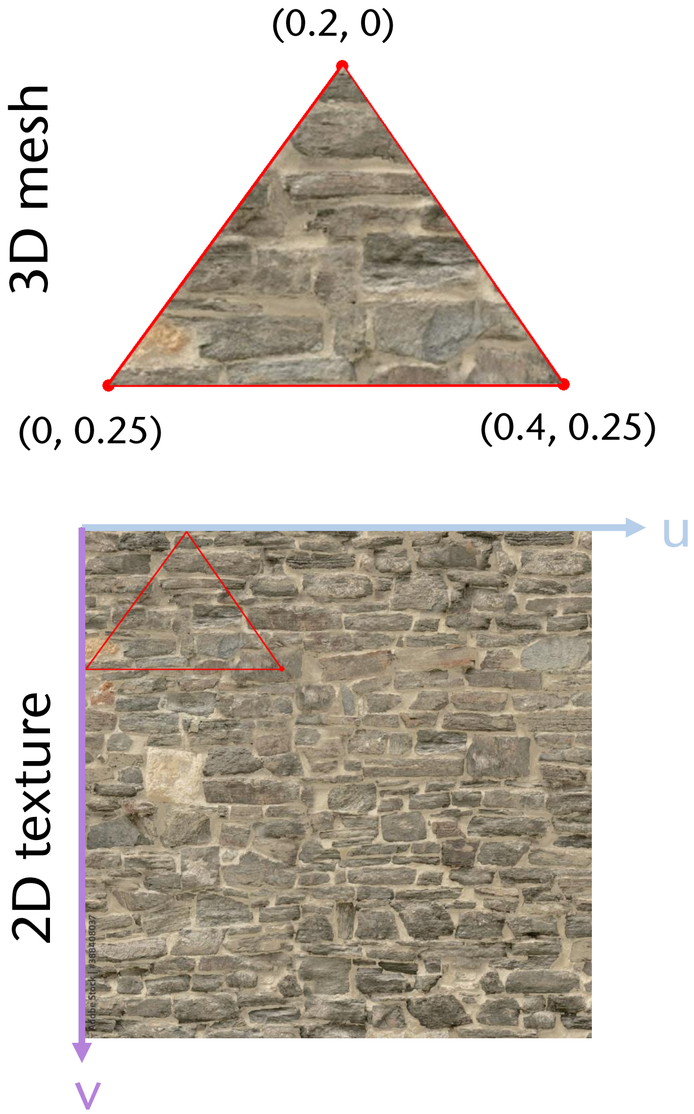
- Vertices store attributes (“Vertex attributes”):
  - Position
  - Normal
    - Here, it points towards the screen (negative y axis)
  - UV coordinates
    - Coordinates in the 2D texture

Recap: Vertex / Vertices



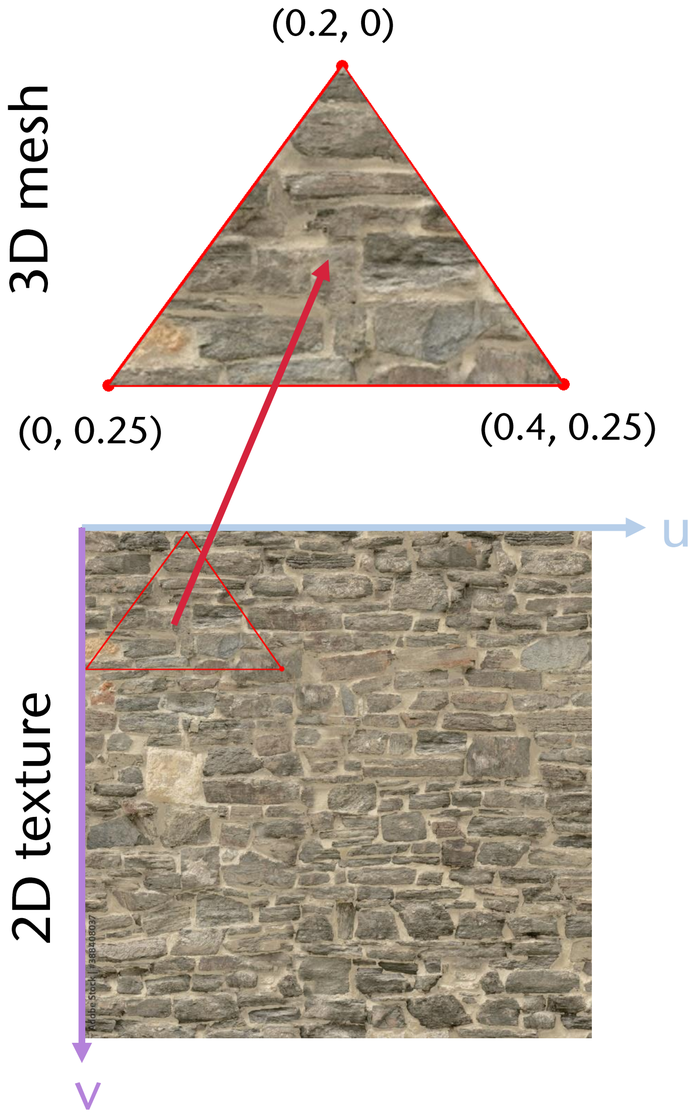
- Vertices store attributes (“Vertex attributes”):
  - Position
  - Normal
    - Here, it points towards the screen (negative y axis)
  - UV coordinates
    - Coordinates in the 2D texture (≈ 2D image)

Recap: Vertex / Vertices



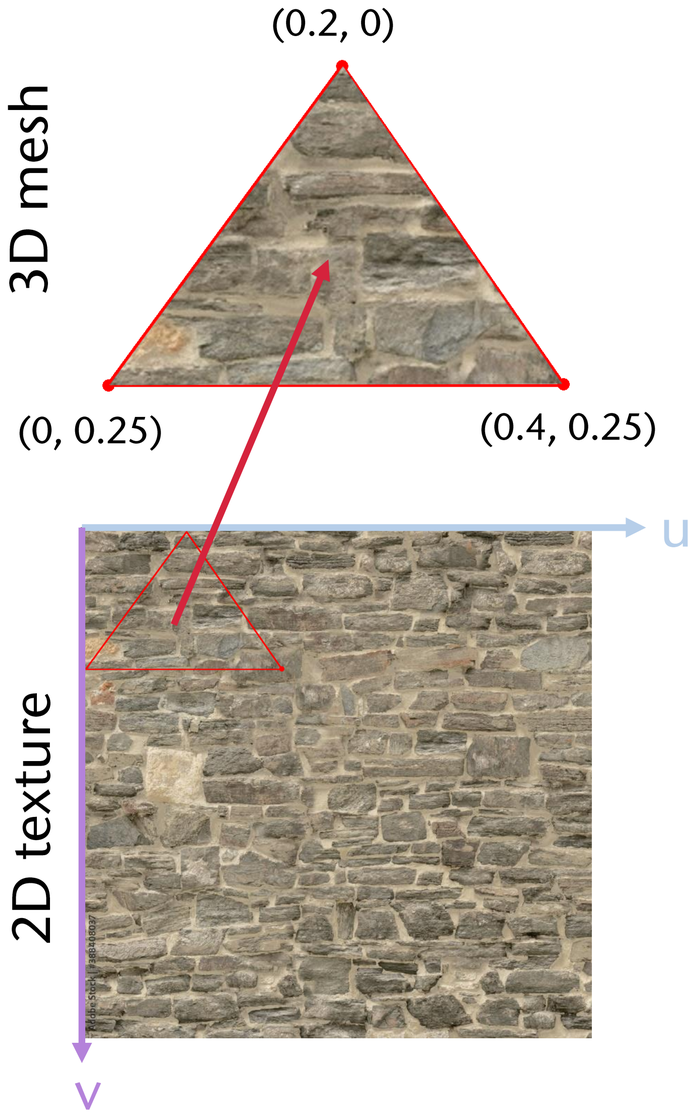
- Vertices store attributes (“Vertex attributes”):
  - Position
  - Normal
    - Here, it points towards the screen (negative y axis)
  - UV coordinates
    - Coordinates in the 2D texture (≈ 2D image)

Recap: Vertex / Vertices



- Vertices store attributes (“Vertex attributes”):
  - Position
  - Normal
    - Here, it points towards the screen (negative y axis)
  - UV coordinates
    - Coordinates in the 2D texture (≈ 2D image)

Recap: Vertex / Vertices



- Vertices store attributes (“Vertex attributes”):
  - Position
  - Normal
    - Here, it points towards the screen (negative y axis)
  - UV coordinates
    - Coordinates in the 2D texture (≈ 2D image)
    - Only 2D, since this are coordinates in the 2D texture

Recap: Faces

- Faces often store:
  - References to vertices
  - Which material to use

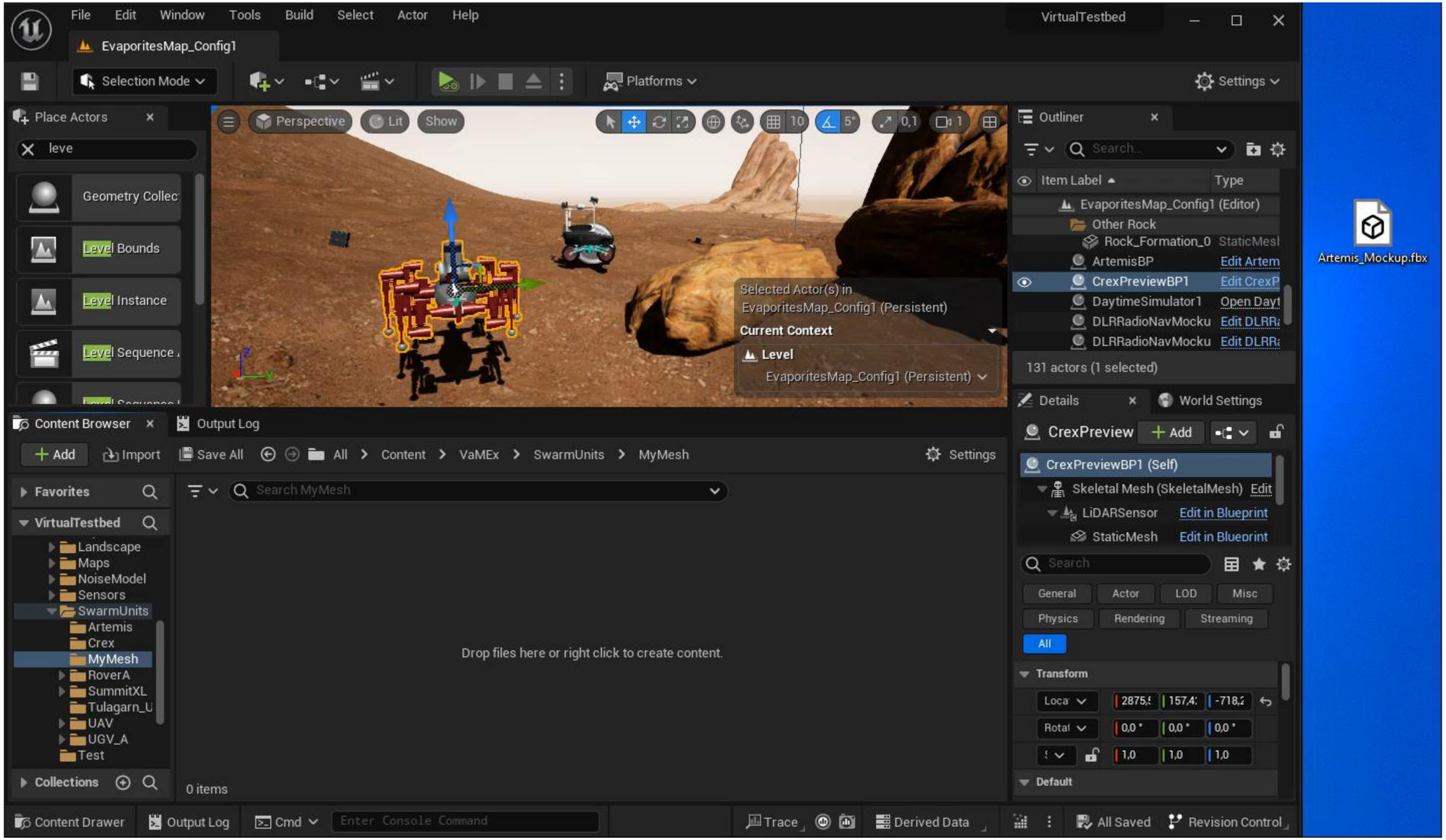
- Edges often not explicitly defined
  - ➢ Results from the face definition

Meshes in Unreal Engine 5

Meshes in Unreal

- 3D-Meshes can be modelled in Blender, Maya, etc.
  - These meshes can be exported as **.obj** or **.fbx** file
  - These **.obj** or **.fbx** files can be dragged into a folder in the Content Browser
  - After import, these files are stored as **.uasset** file in your project
  - Then, you can attach that StaticMesh to an StaticMeshComponent

Import an .fbx file



StaticMesh vs. SkeletalMesh

- Unreal has different types of Meshes
- The two most important mesh types are:
  - A StaticMesh:
    - This is the typical mesh we know.
  - A SkeletalMesh:
    - Is the typical mesh with an additional skeleton.
    - A deformable mesh whose individual parts can be animated separately
      - e.g. people (with arms and legs), animals, etc.

StaticMesh vs. SkeletalMesh

- Motivation for a Skeleton:
  - Human meshes often have many vertices
  - Animation without skeleton:
    - **Problem:** We have to say where each vertex has to be at time ***t***.
      - This would take a very long time for the animator, since there is often a huge amount of vertices in an object

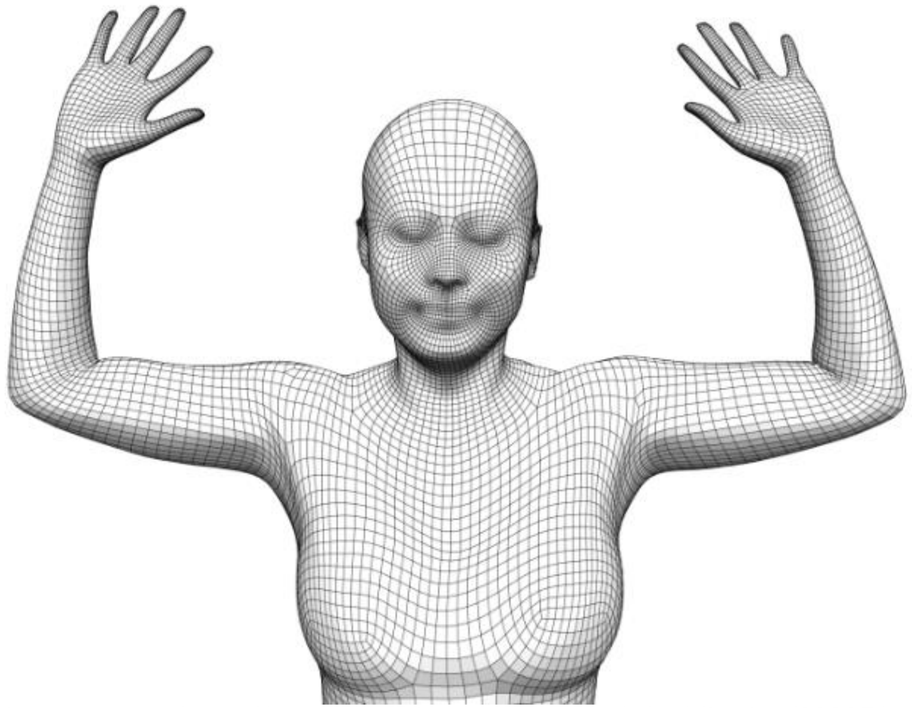


Image source: https://www.graphics.rwth-aachen.de/publication/03337/

StaticMesh vs. SkeletalMesh

- Motivation for a Skeleton:
  - Instead, we create a **skeleton** consisting of **bones**.
    - Creating a skeleton is known as “rigging”.
  - We map bones to vertices with an impact weight
    - Also known as “skinning” and “weight painting”
    - This is done in Blender or Maya, not the Unreal Engine!
  - **Result:** We only have to animate a few bones, instead of the large amount of vertices!

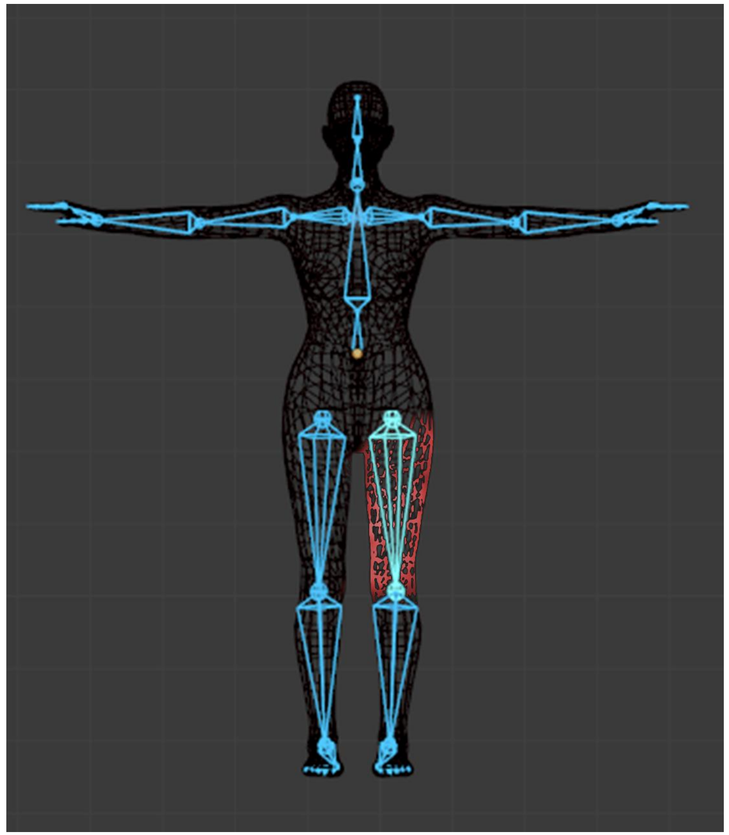


Image Source: https://docs.blender.org/manual/en/latest/animation/armatures/skinning/introduction.html

StaticMesh vs. SkeletalMesh

- Why is this important?
  - Unreal uses skeletal meshes for characters in the default template.
  - A static mesh is **not** compatible with a skeletal mesh.
  - You want to use **static** meshes most of the time.

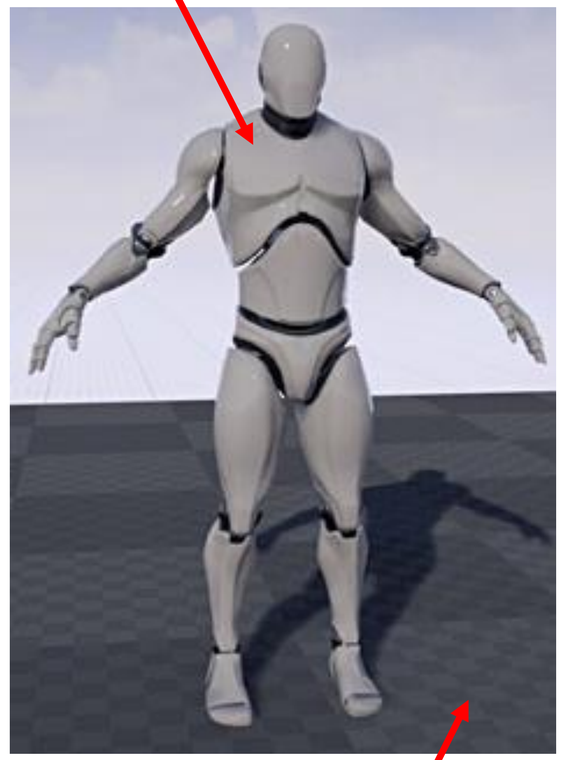
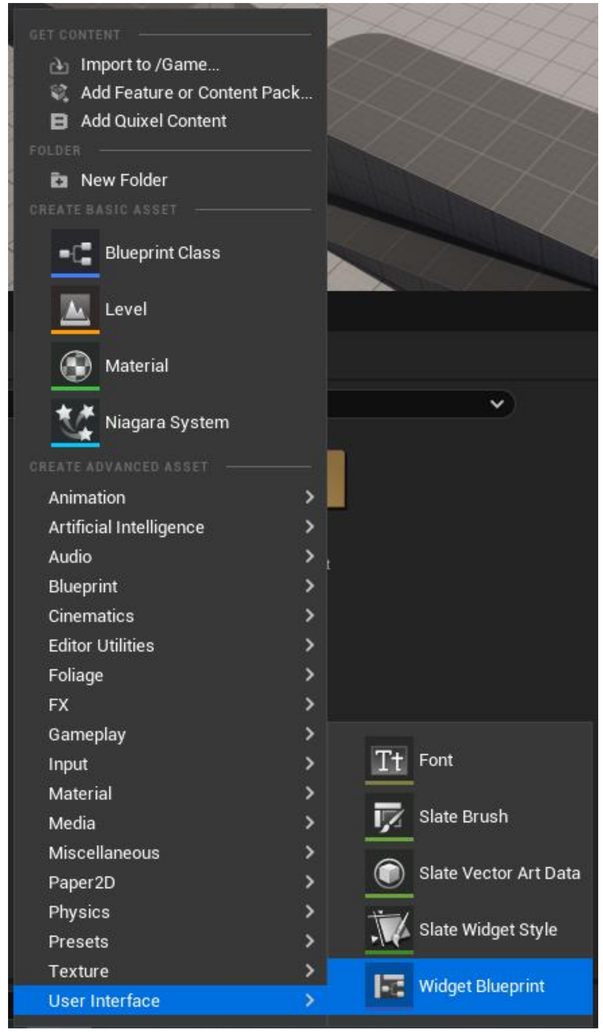


Image Source: https://forums.unrealengine.com/t/how-to-add-the-deafult-player-model-with-animations-to-an-empty-project-with-starter-content/409906

GUI

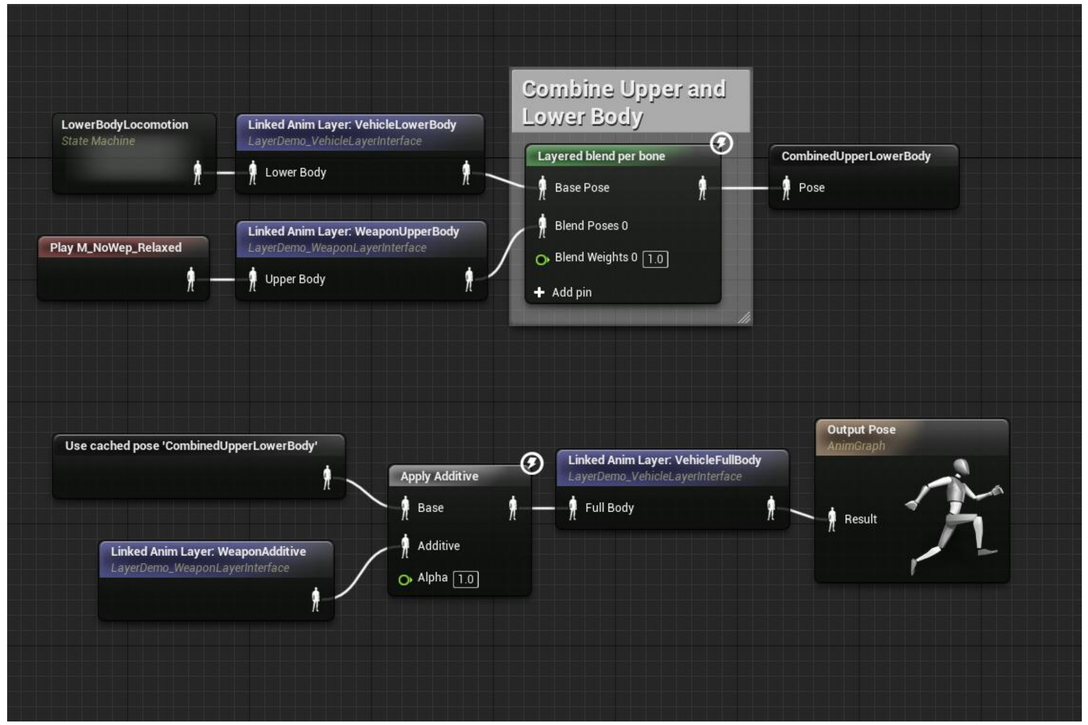
- Unreal has a Widget Blueprint
  - This Widget can be designed and programmed directly in the Unreal Editor via Blueprints

Image Source: https://blog.gamedev.tv/creating-unreal-engine-ui-with-umg-and-c/

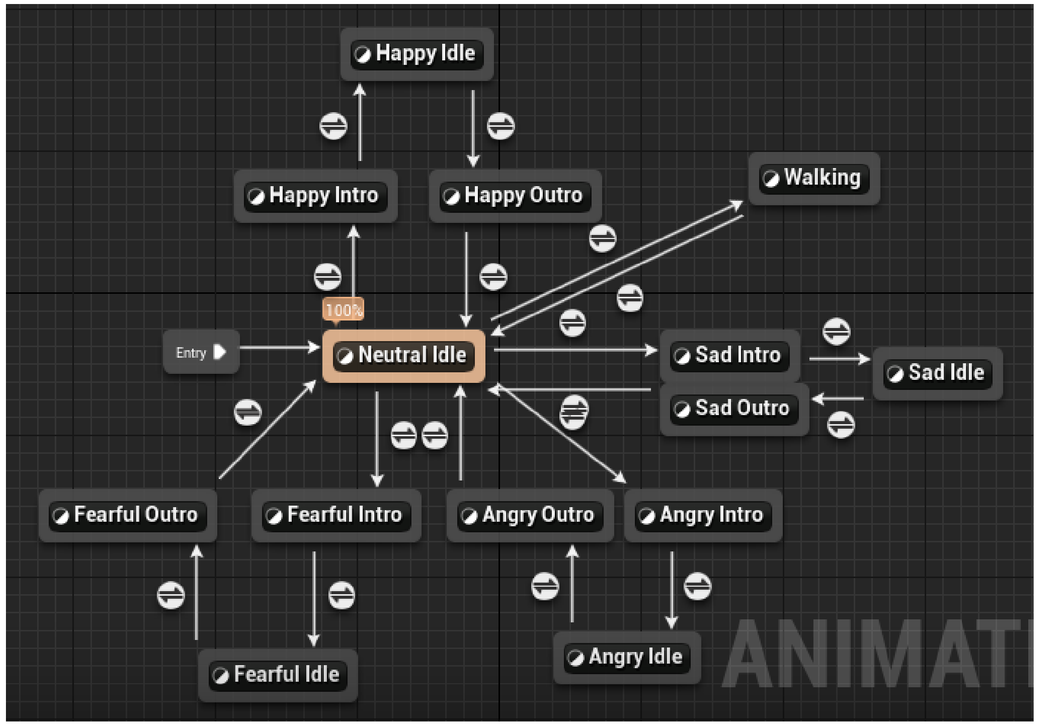


Animation

- Animation Blueprints
  - Allows the creation of complex animation for **Skeletal**Meshes



https://docs.unrealengine.com/5.0/en-US/using-animation-blueprint-linking-in-unreal-engine/



https://docs.inworld.ai/docs/tutorial-integrations/unreal-engine/gallery/animation/

Remarks

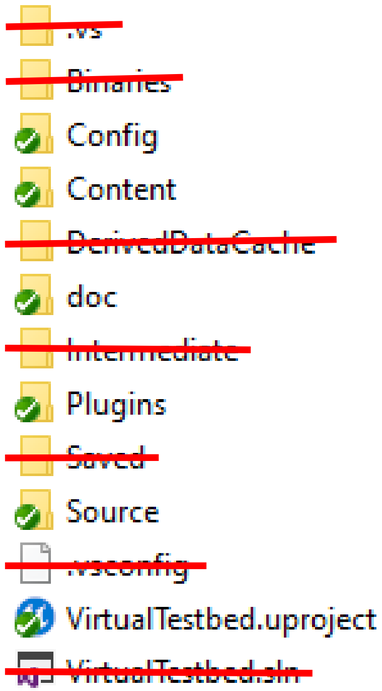
- Learning material:
  - Official Unreal Engine Documentation:
    - https://docs.unrealengine.com/5.3/en-US/
  - YouTube videos:
    - For almost everything there is a "HowTo" video on YouTube.
    - Not necessarily always "best practice", but often very good.

Remarks

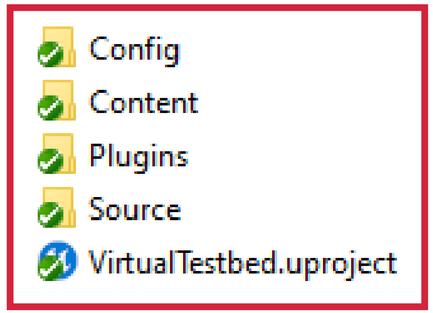
- Unreal Engine Projects tend to get pretty huge (in terms of hard disk space)
  - When 20 groups are submitting a project in **Media Engineering** or **Virtual Reality** course, this requires extremely much space for tutors and the infrastructure...
  - The solution is (on the next page)...

Remarks

- Remove unneeded folders and files before submitting your project!



- To reconstruct an unreal project, not all folders and files are required!
- Please delete them before submitting!
- When using git, you should setup a .gitignore file for your project.



These 4 folders and the uproject file are enough to open your full project!