Voice from Communities

- Currently Programming Assignment 3 is still open for submission. We have received more than 400 submissions. After our teaching assistants finished reviewing all the submissions, we will sort the problems in them and provide answers. Meanwhile, we are making the FAQ manual to help your studying progress.

- To address overdue submissions, we will reopen Programming Assignment submission later. Practicing is the fastest way for studying. We hope no matter when you enter the course, you can join the practice in Programming Assignments. Come to practice and discuss in the community!

Q&A about Piccolo Engine

- Q1: Why does Piccolo Engine use custom asset format instead of universal formats such as glTF or FBX?
    - DCC formats are not suitable for game engines
    - Data organization needs to be optimized for runtime performance
    - Many formats are not open-sourced. SDKs are required to be integrated to use them
- Q2: Why does Piccolo Engine use MoltenVK on macOS instead of Metal API?
    - MoltenVK maps Vulkan to native Metal API on macOS
    - Currently we use Vulkan for cross-platform compatibility
    - Native graphic APIs will be supported in the future
- Q3: When does Programming Assignment 3: animation and physics due?
    - 18 July 23:59:59

Lecture 14

Tool Chains

Applications & Advanced Topic

WANG XI

GAMES 104

2022

Outline of Tool Chains

01.

Foundation of Tool Chains

- What is Game Engine Tool Chains
- Complicated Tool GUI
- How to Load Asset - Deserialization
- How to Make a Robust Tools
- How to Make Tool Chain
- What You See is What You Get
- One More Thing - Plugin

02.

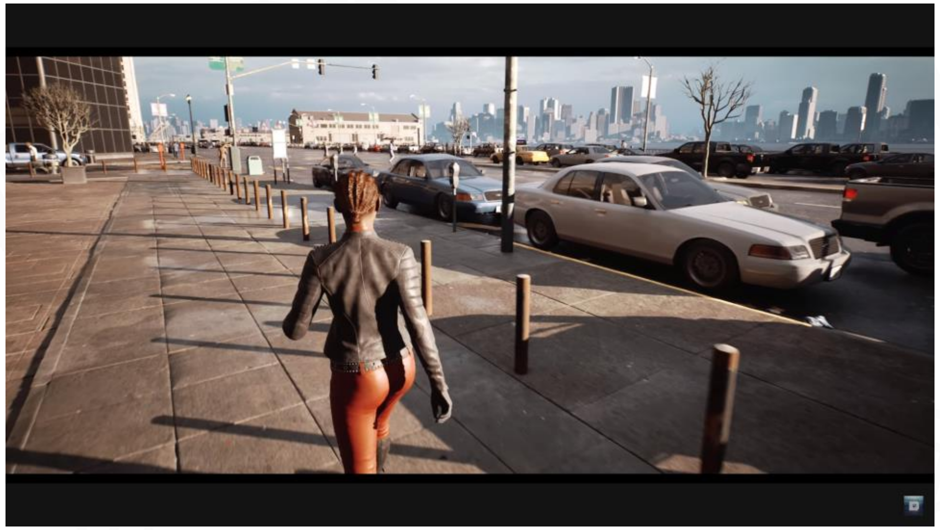
Applications & Advanced Topic

- Glance of Game Production
- Architecture of A World Editor
- Plugin Architecture
- Design Narrative Tools
- Reflection and Gameplay
- Collaborative Editing

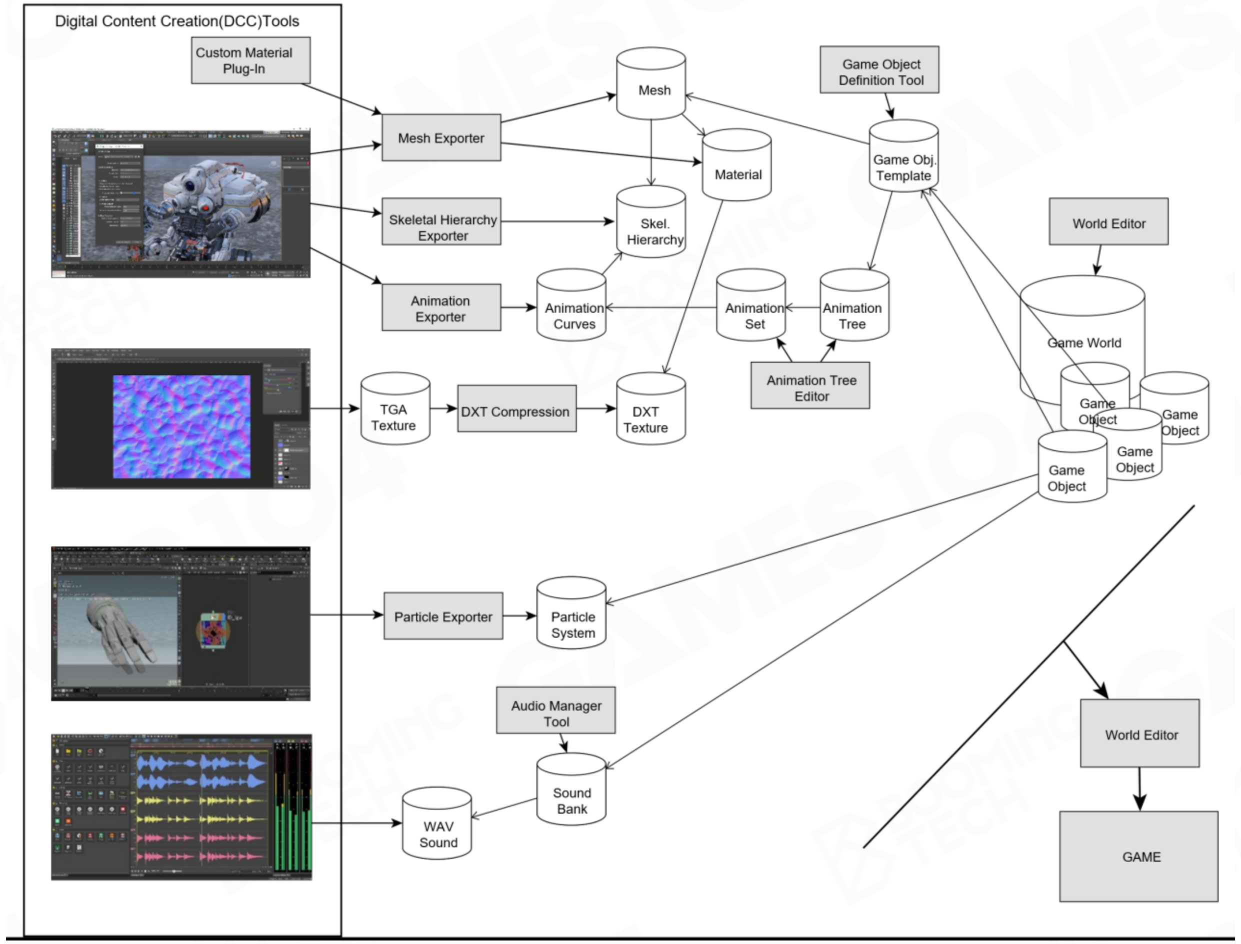
Glance of Game Production

whether to work on the environment, the animation, to place characters or to create missions.

Adapt to Different Game Genres



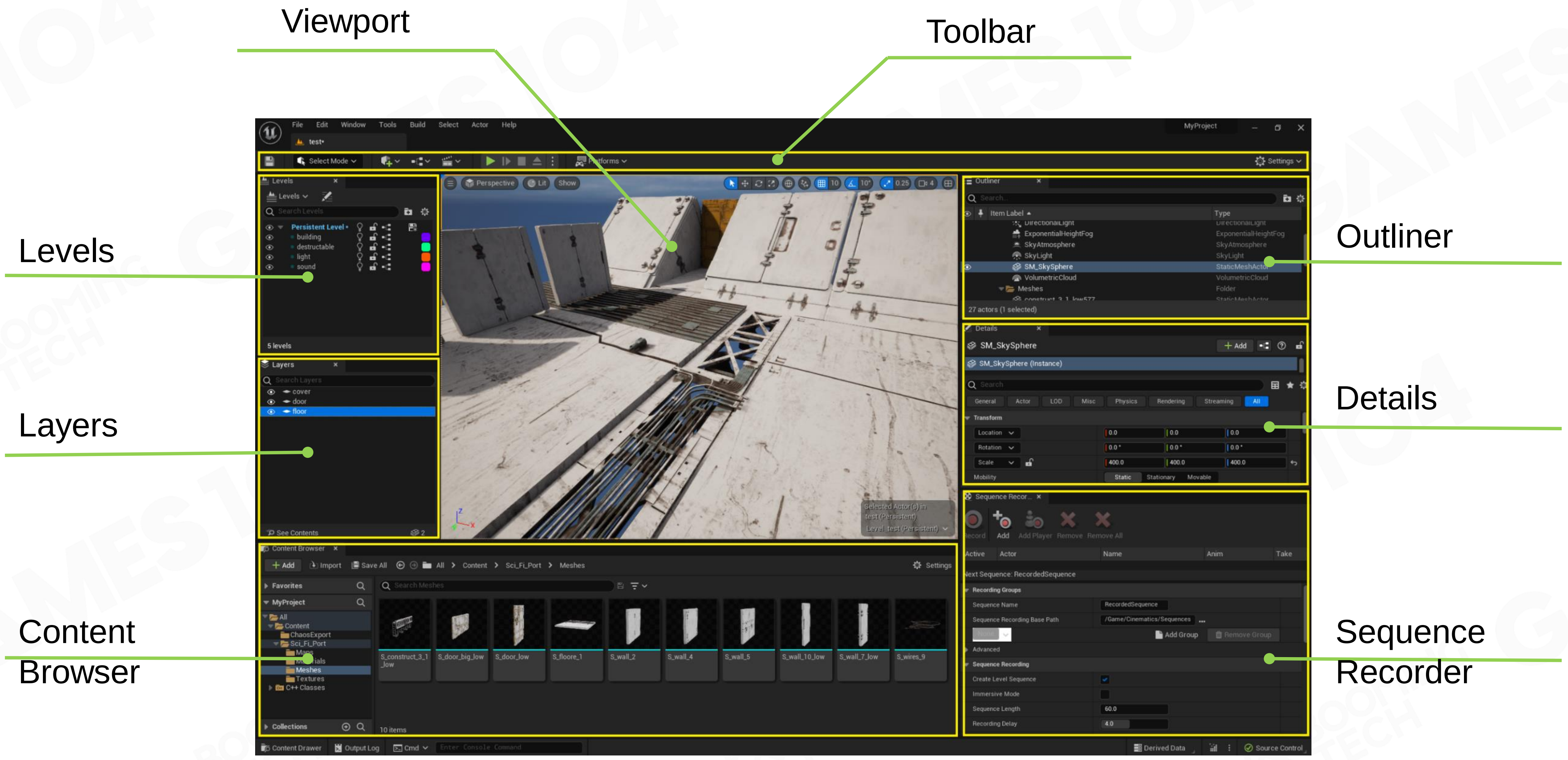
Challenges from Real Production



- Massive various data from DCC and engine tools
- Artist, designer and programmer with different mindsets
- WYSIWYG is must for high quality production

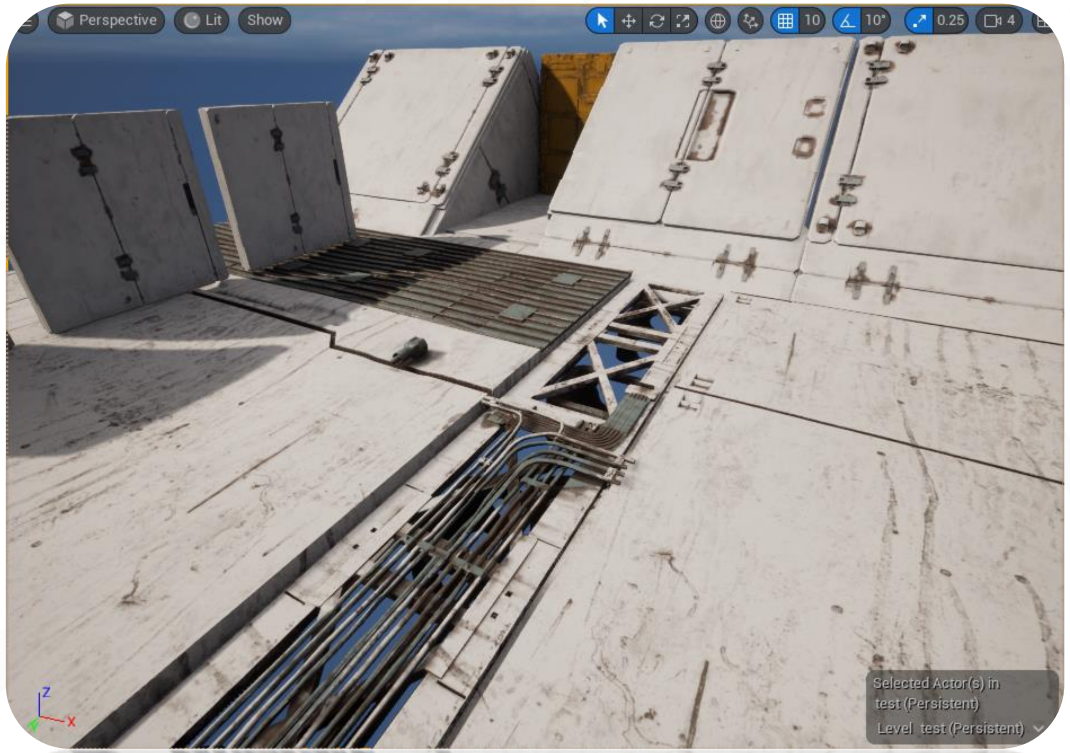
World Editor

- A hub for everything to build the world


Viewport
Toolbar
Levels
Outliner
Details
Layers
Content Browser
Sequence Recorder

Editor Viewport : A Special Version of Game Engine

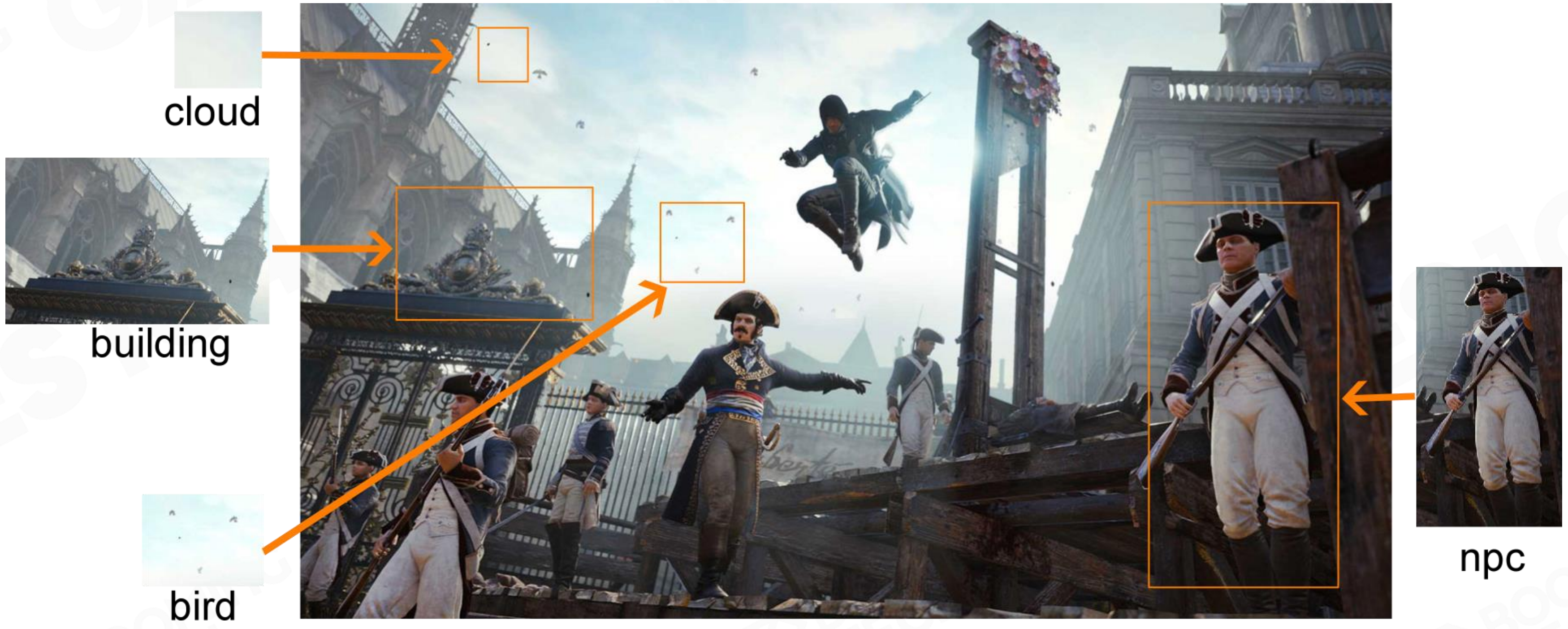
- Main window of interaction between designers and game world
- Powered by a full game engine in special “editor” mode
- Provides a variety of special gadgets and visualizers for editing



WARNING: Editor-only code must be moved out of released game!

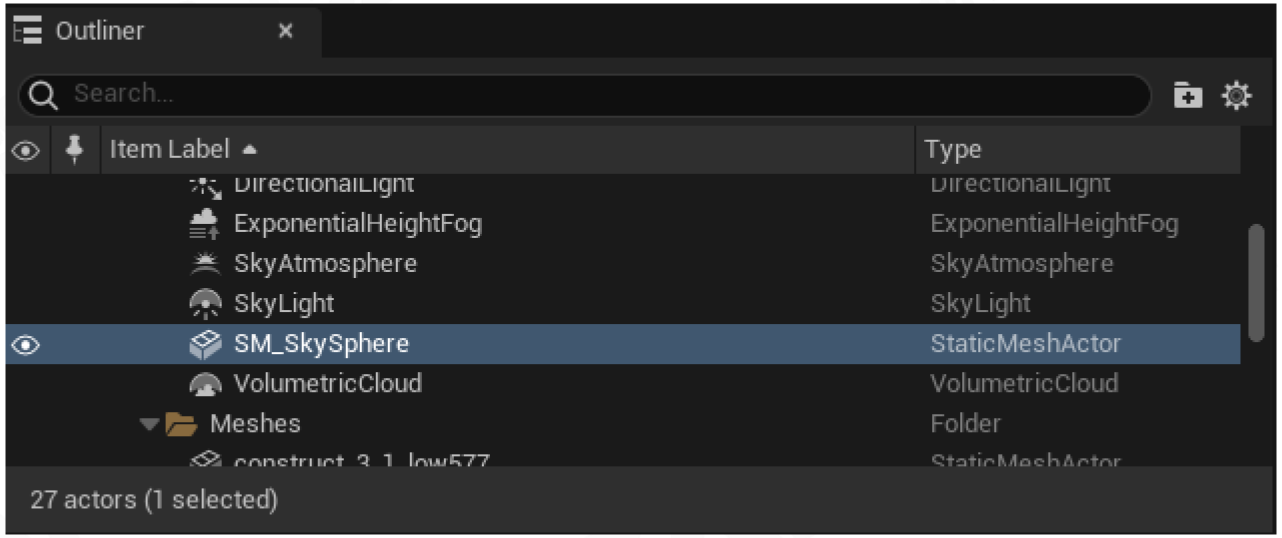
Everything is an Editable Object

- The editing requirements of all objects in the editor world are mostly the same, such as moving, adjusting parameters, etc



Different Views of Objects

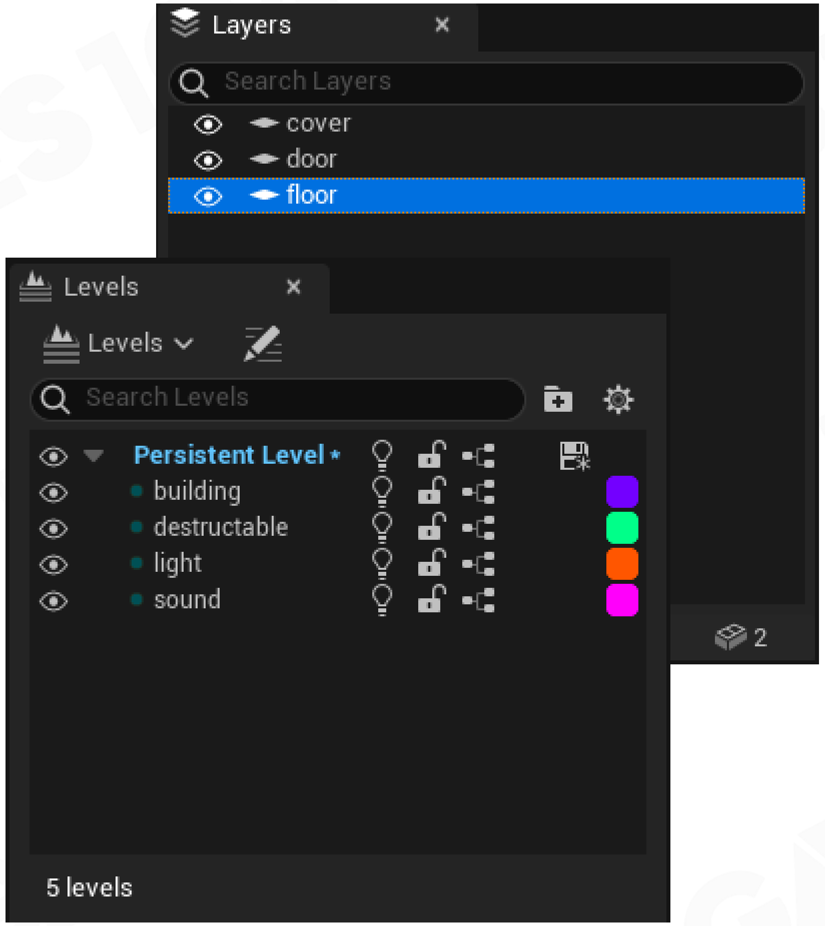
- Display all of the objects within the scene
- Organize objects in different views for user conveniences



Item Label	Type
DirectionalLight	DirectionalLight
ExponentialHeightFog	ExponentialHeightFog
SkyAtmosphere	SkyAtmosphere
SkyLight	SkyLight
SM_SkySphere	StaticMeshActor
VolumetricCloud	VolumetricCloud
Meshes	Folder
construct_3_1_low577	StaticMeshActor

27 actors (1 selected)

Tree view



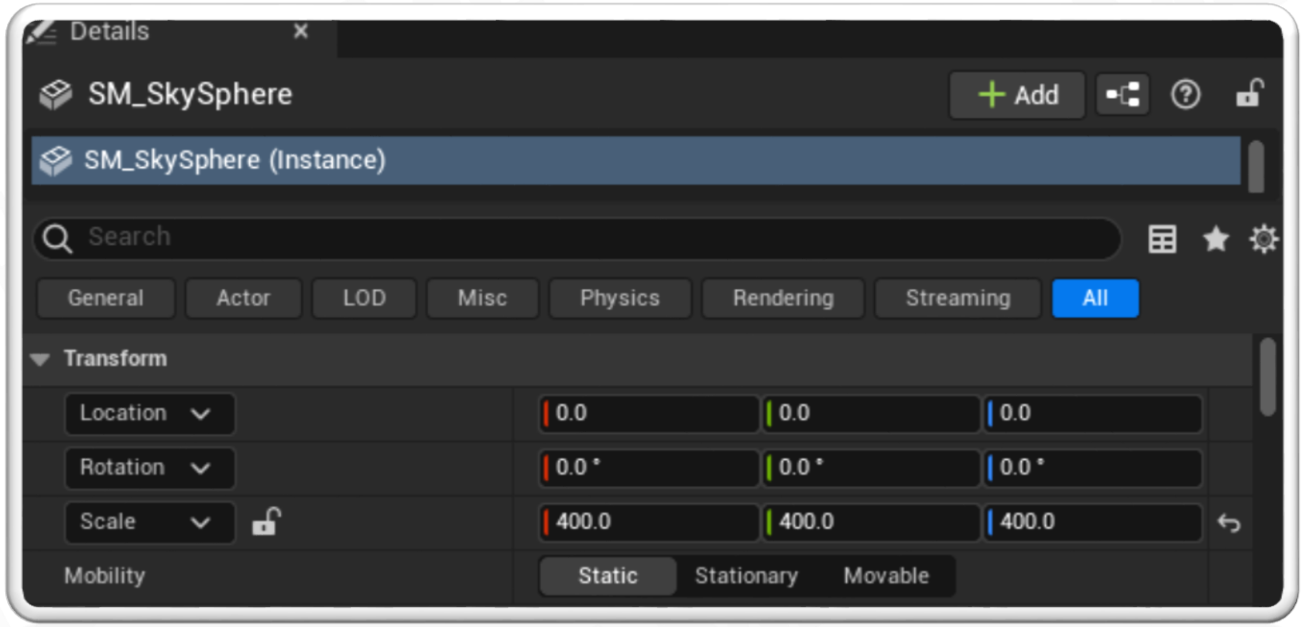
Persistent Level *
building 🟪
destructable 🟩
light 🟧
sound 🟪

5 levels

Categories and groups

Schema-Driven Object Property Editing

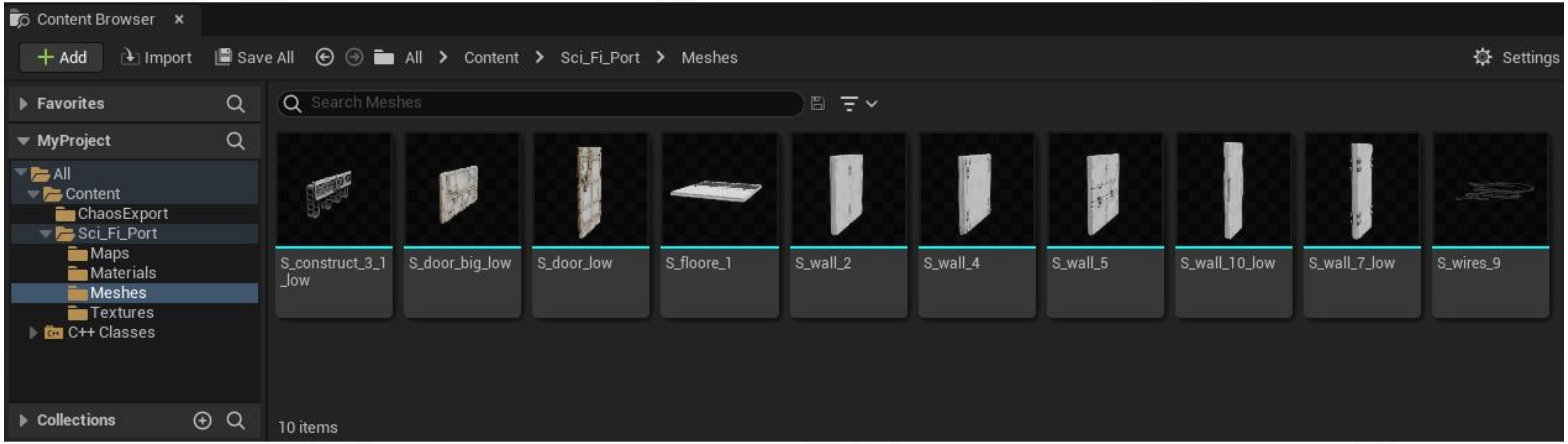
- Displays all of the editable properties for the selected objects
- Beyond schema, we can define some customized editing utilities for different types



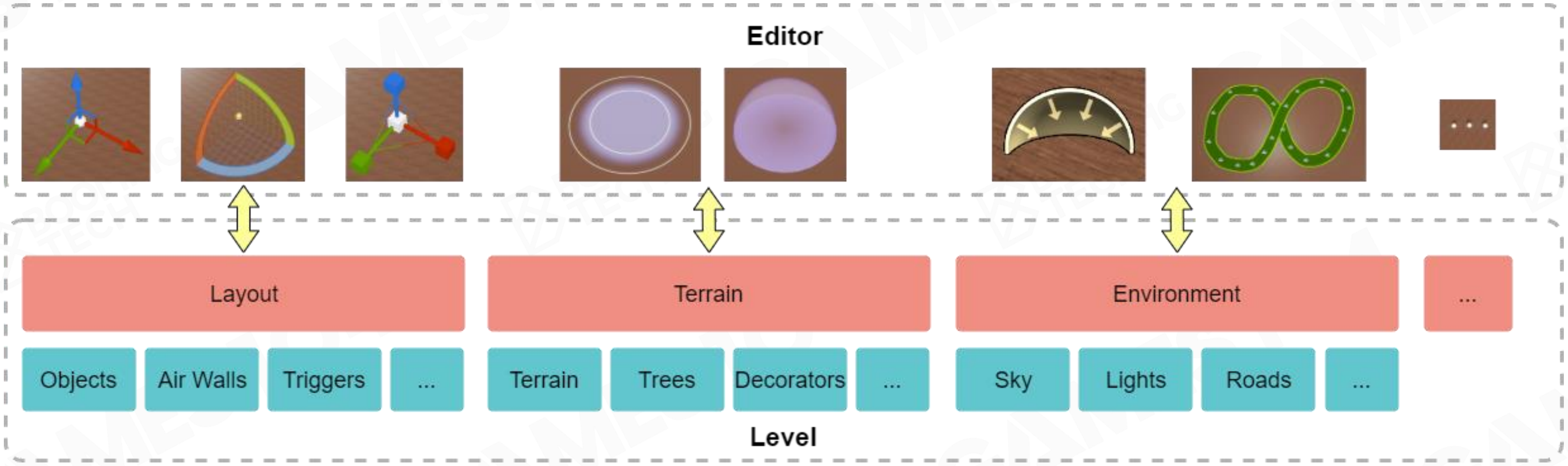
Location 0.0 0.0 0.0
Rotation 0.0 ° 0.0 ° 0.0 °
Scale 400.0 400.0 400.0
Mobility [✓ Static] Stationary Movable

Content Browser

- Provide intuitive thumbnail of all assets
- Share asset among different projects
- Evolution of asset management from static file folder to content "ocean"



Editing Utilities in World Editor



Mouse Picking

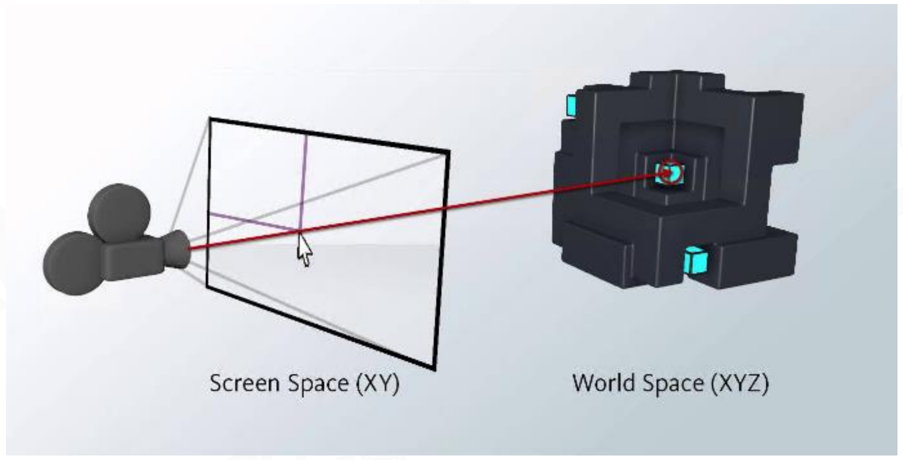
Ray Casting

Pros:

- No cache required
- Can support multiple objects on selected rays

Cons:

- Poor query performance



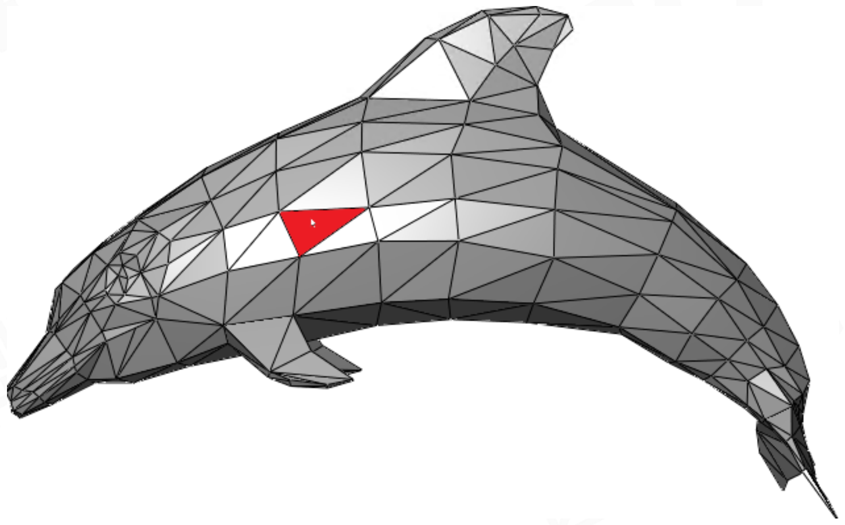
RTT

Pros:

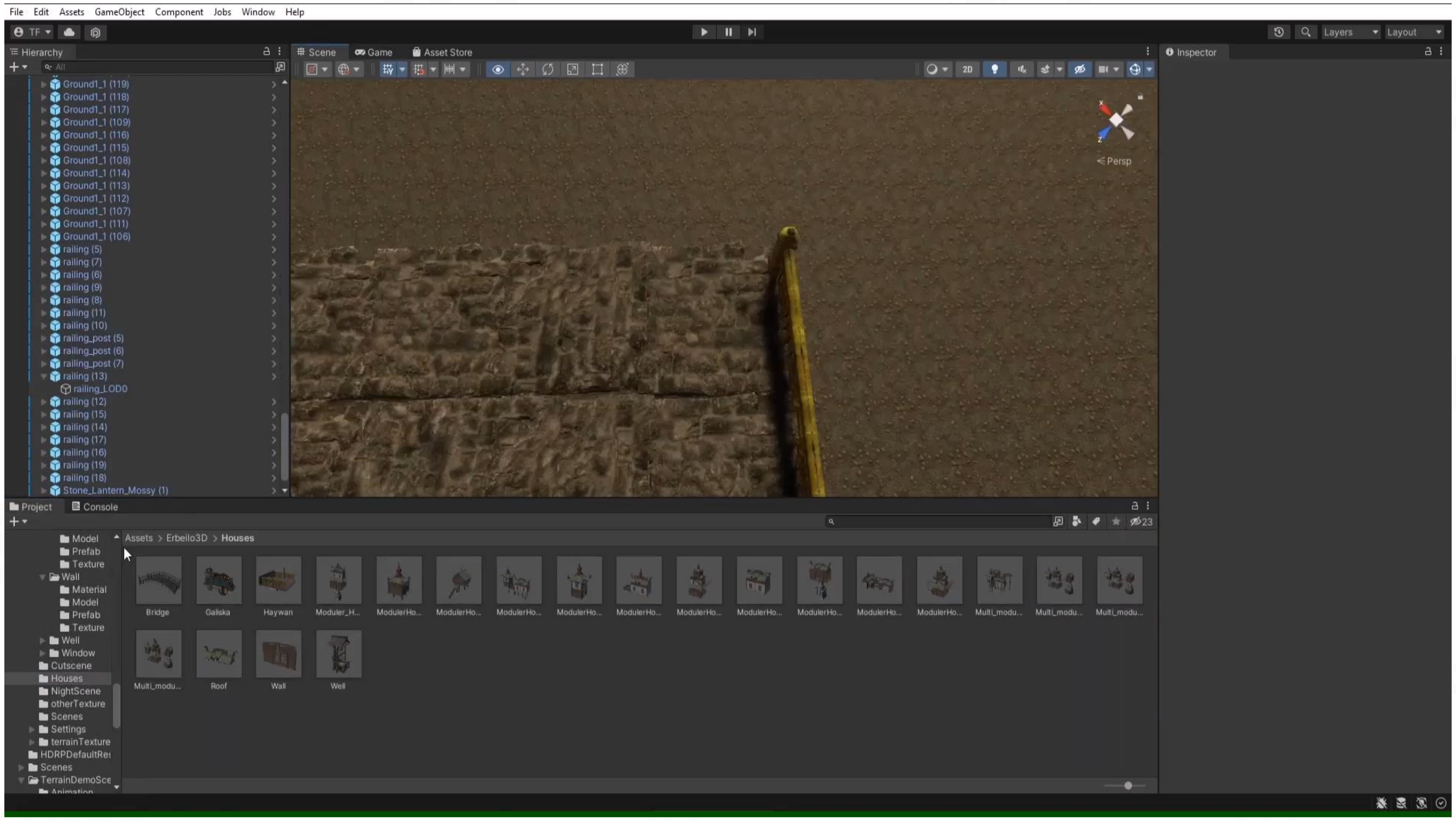
- Easy to implement range queries
- Ability to complete queries quickly

Cons:

- Need to draw an extra picture
- Obstructed objects cannot be selected



Object Transform Editing



Terrain

Landform

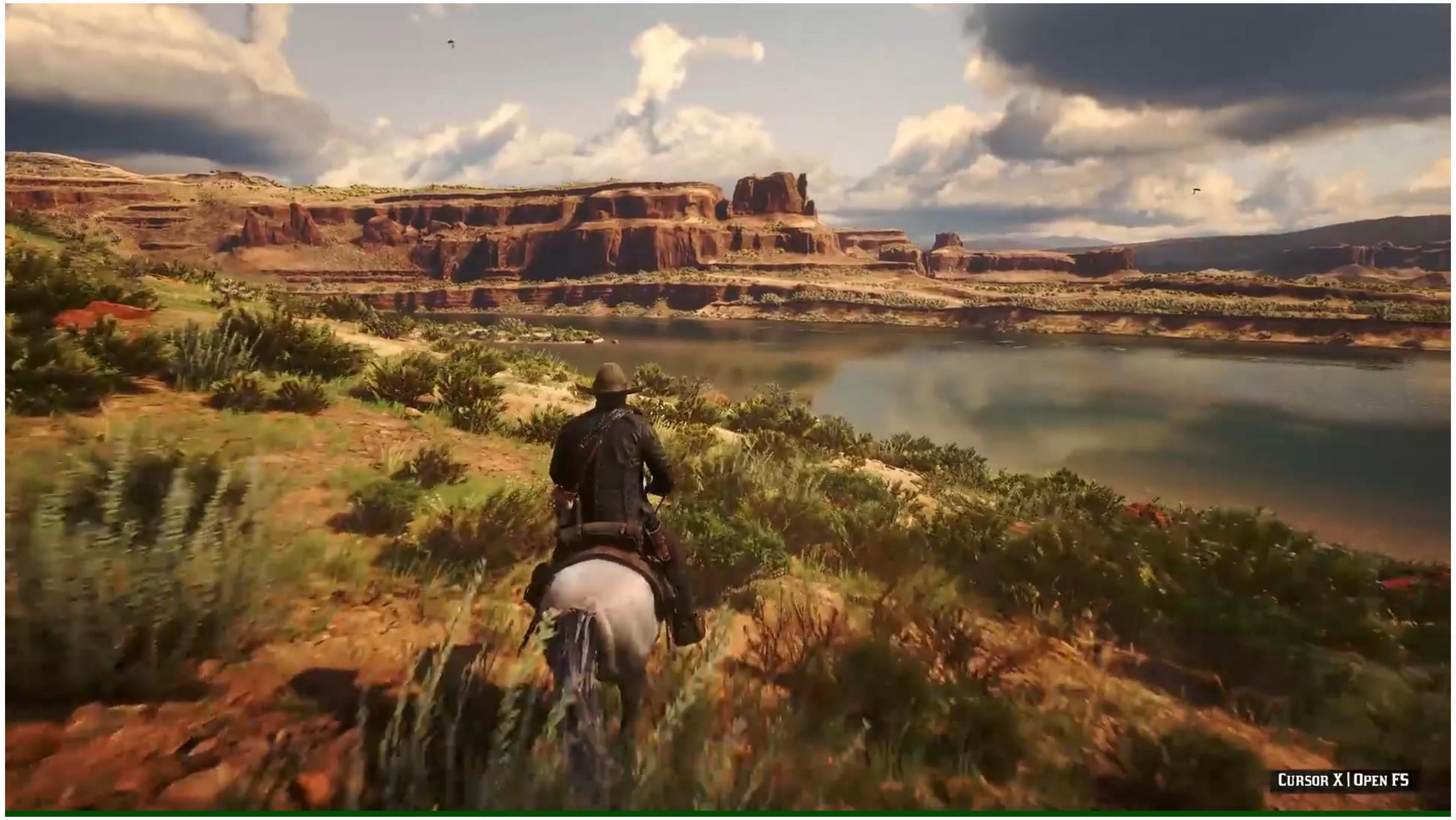
- Height map

Appearance

- Texture map

Vegetation

- Tree instances
- Decorator distribution map



Height Brush

- Draw height map to adjust terrain mesh
  - Height change needs to be natural and smooth
  - Can be easily adjusted to the desired results
    - Customized brush

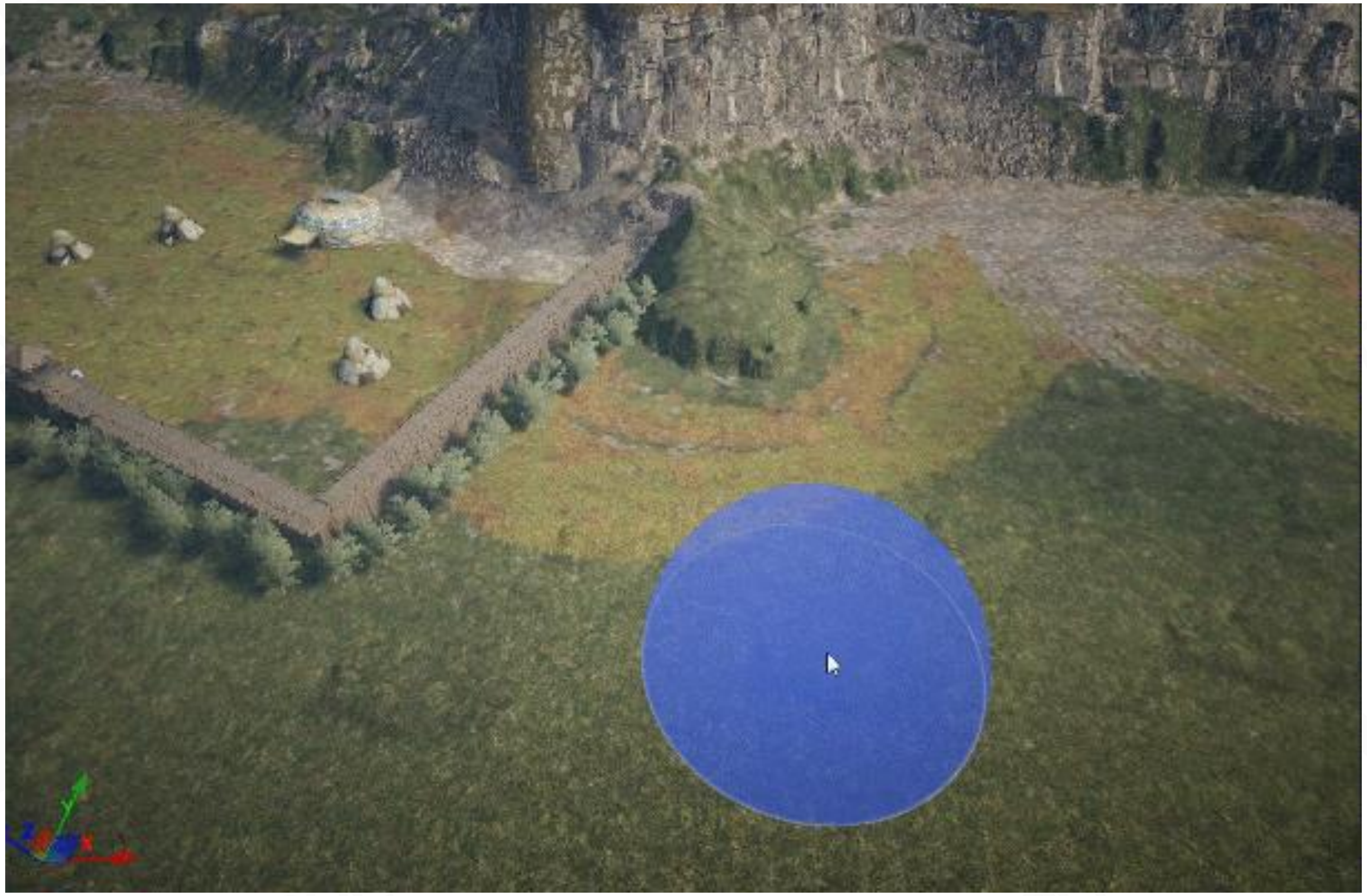
Instance Brush

Pros:

- Instance position is fixed
- Available to further modification

Cons:

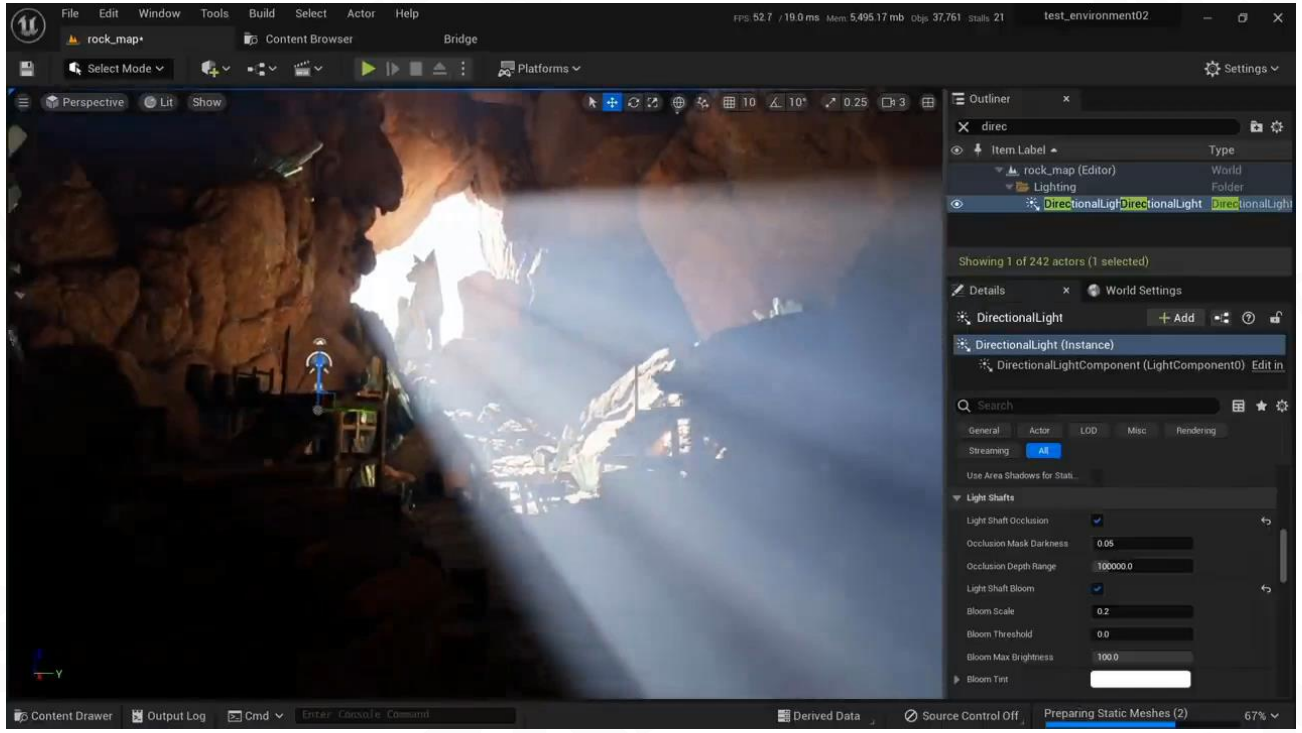
- Large amount of data



Environment

- Sky
- Light
- Roads
- Rivers
- …

From up to down, environment around us present a live world to the player. Edit these environment elements would also be important.



direc

Item Label	Type
rock_map (Editor)	World
Lighting	Folder
DirectionalLightDirectionalLight	DirectionalLight

Showing 1 of 242 actors (1 selected)

DirectionalLight
DirectionalLight (Instance)
DirectionalLightComponent (LightComponent0) Edit in

Use Area Shadows for Stati...

Light Shafts

Light Shaft Occlusion	✓
Occlusion Mask Darkness	0.05
Occlusion Depth Range	100000.0
Light Shaft Bloom	✓
Bloom Scale	0.2
Bloom Threshold	0.0
Bloom Max Brightness	100.0
Bloom Tint	⬜

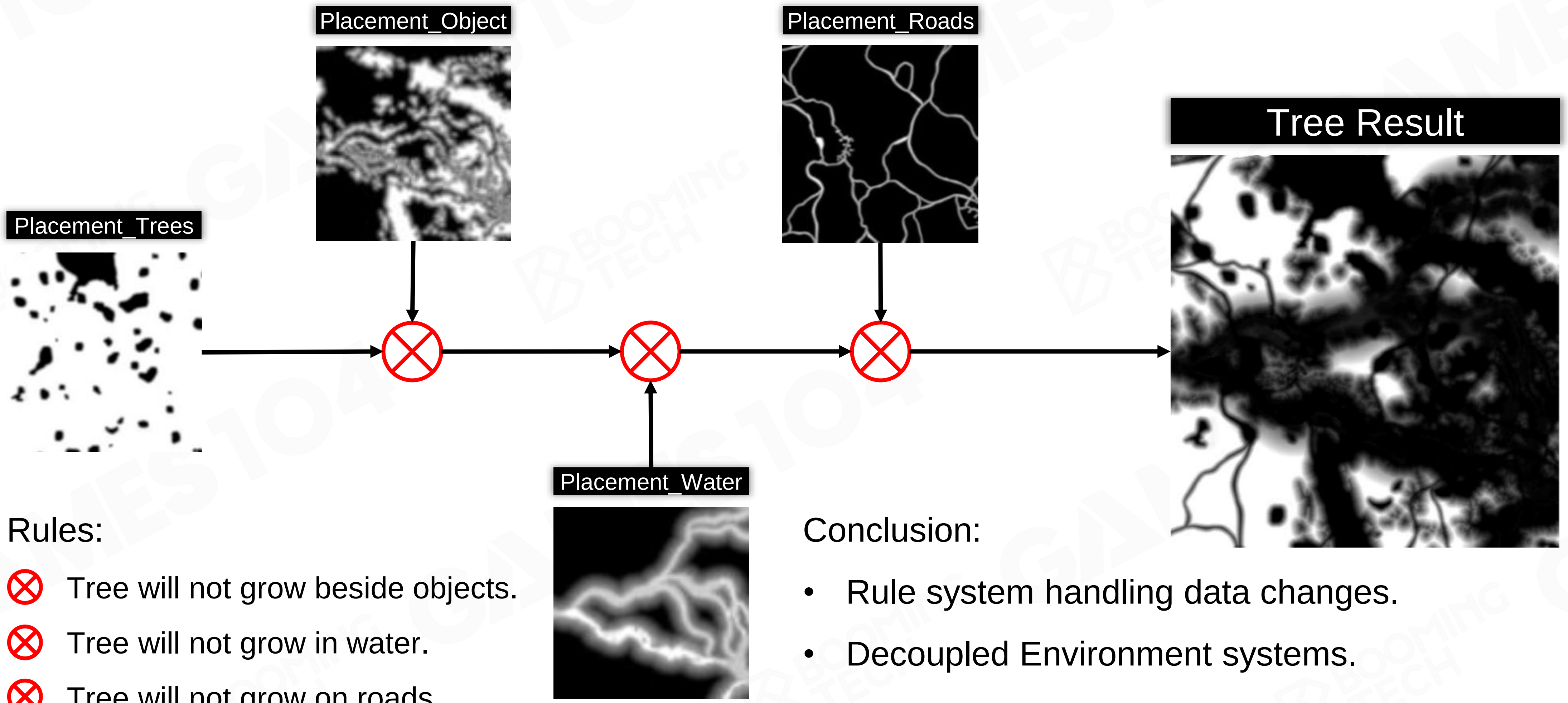
Rule System in Environment Editing

There are various realistic rules when we build the environment:

- No trees on the road.
- No decorators on the road.
- Road should fit the terrain.
- Stones usually lay on road sides.
- …

Road system would infect the data of other environment systems.

Environment — Rule System



Rules:

- ⊗ Tree will not grow beside objects.
- ⊗ Tree will not grow in water.
- ⊗ Tree will not grow on roads.

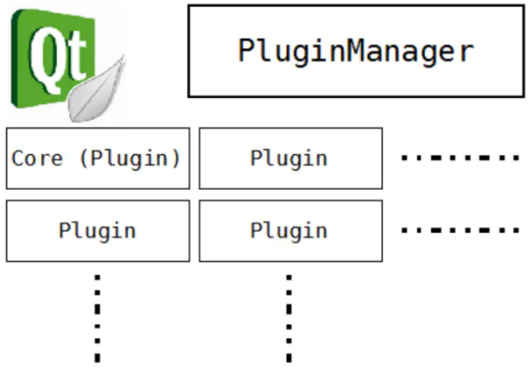
Conclusion:

- Rule system handling data changes.
- Decoupled Environment systems.

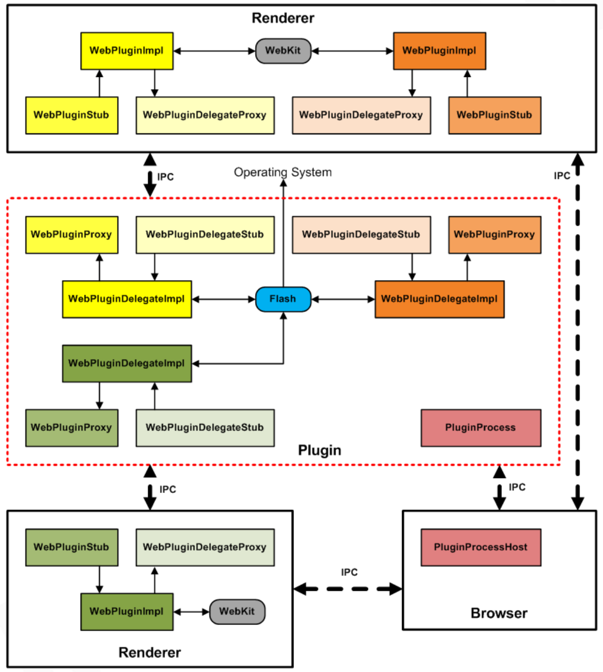
Time: 0.13 Scale: 1 Day: 1 Time: 12:00:06 (Paus
Edit

Editor Plugin Architecture

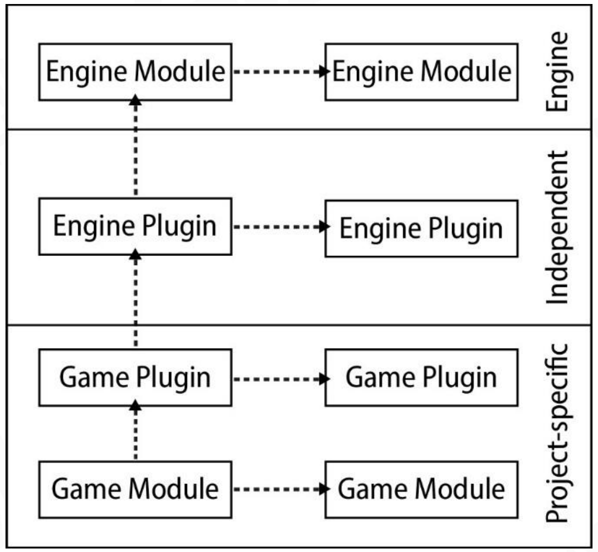
Examples of Plug-in Module in Commercial Softwares



QTCreator

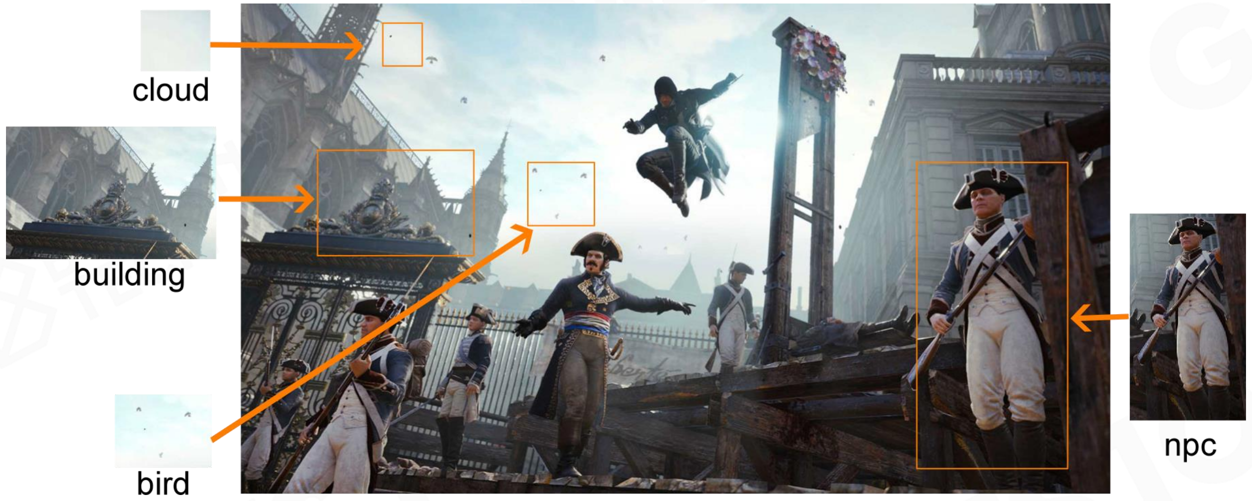


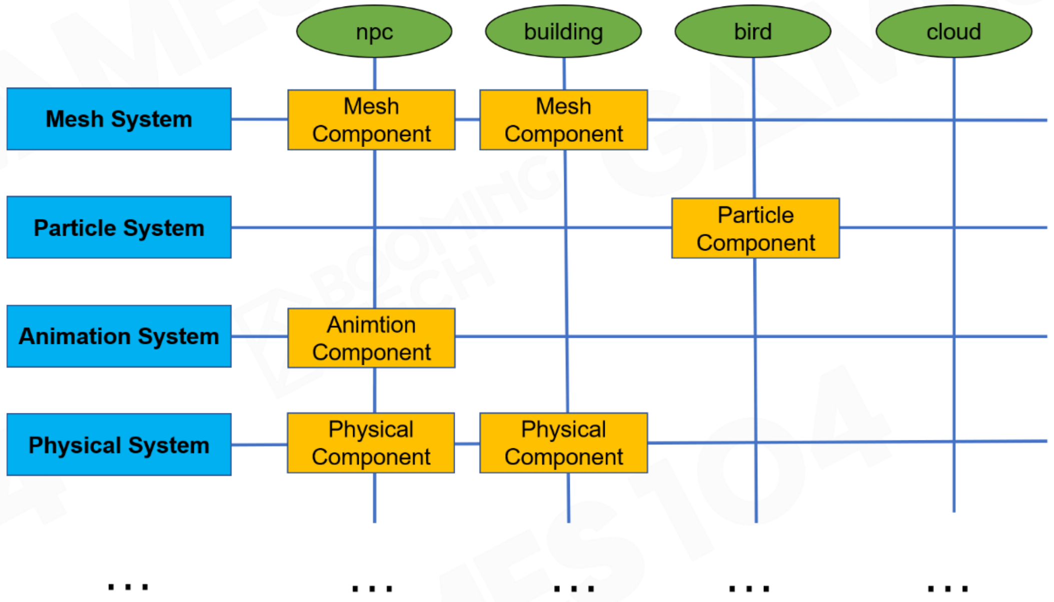
Chrome



Unreal

A Cross Matrix between Systems and Objects



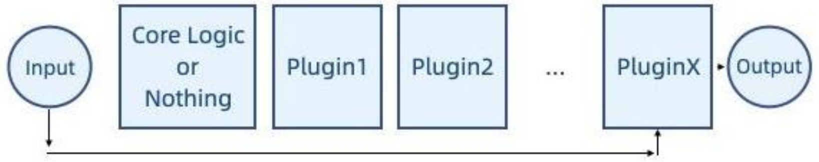


Any system and object type could be plug-ins to Editors

Combination of Multiple Plugins (1/2)

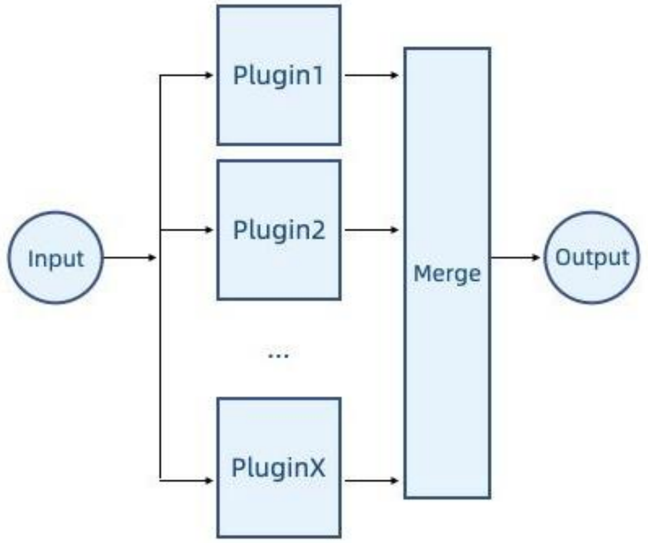
Covered

- Only execute the newly registered logic, skip the original logic
- Ex. Terrain editing overwrite



Distributed

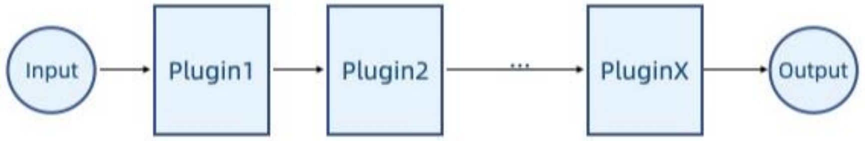
- Each plugin will be executed, and if there is an output, the results will eventually be merged
- Ex. Most special system editing seperately



Combination of Multiple Plugins (2/2)

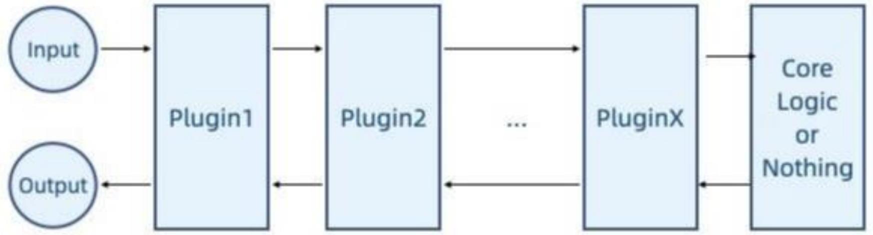
Pipeline

- Input and output are connected to each other, generally input and output are the same data type
- Ex. Asset preprocessing, geometry for physics



Onion rings

- On the basis of the pipeline, the core logic of the system is in the middle, and the plug-in pays attention to the logic of entering and exiting at the same time
- Ex. Road editing plugin with terrain plugin



One More Thing – Version Control

A certain version relationship is required between the plug-in and the host application to ensure that they can work together normally.

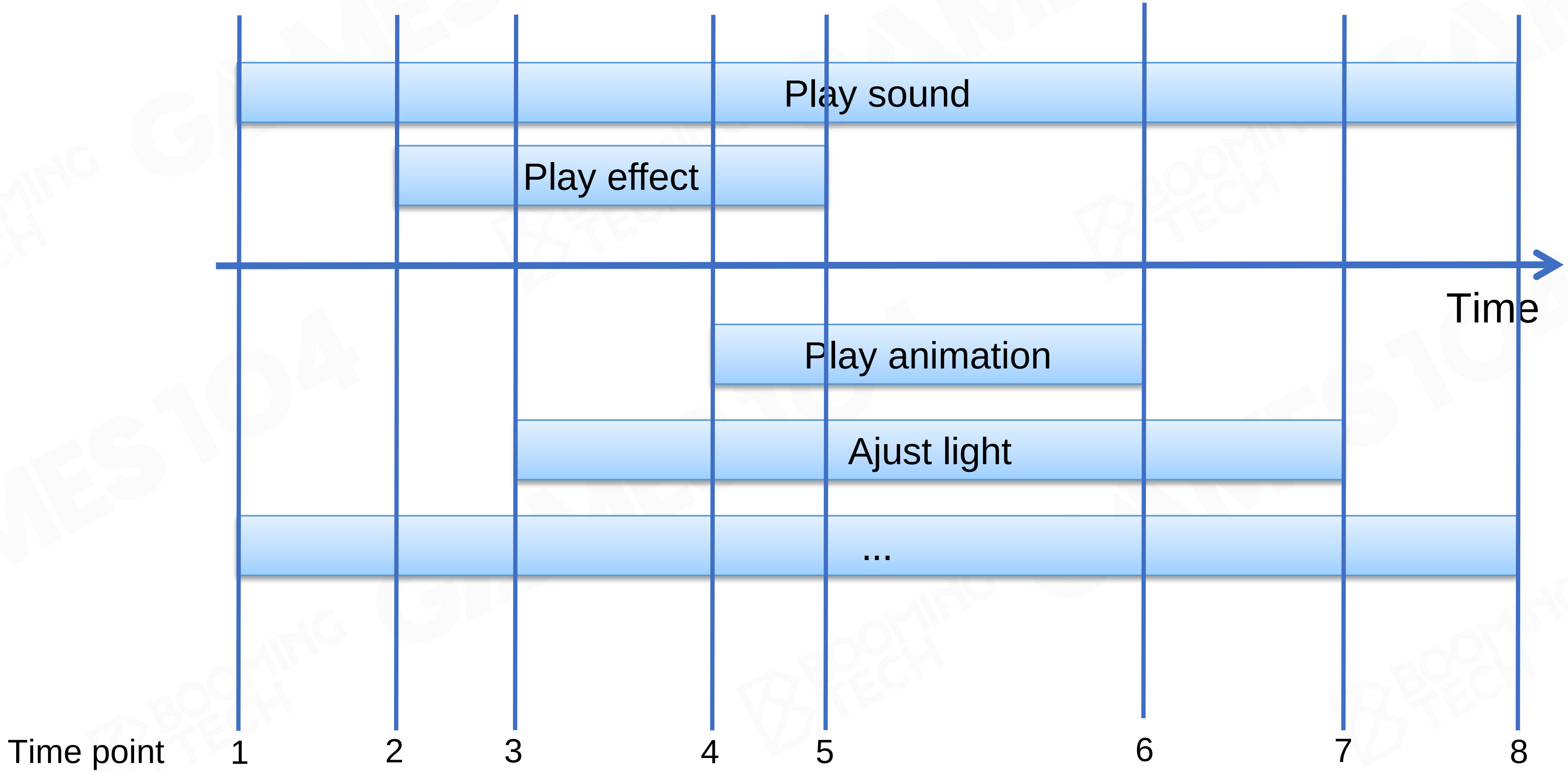
- Plug-in use the same version number with the host application
- Plug-in use the version number of the plug-in interface
  - This is more recommended because the update frequency of the plug-in interface and the software may be different

Design Narrative Tools

Storytelling in Games

Storytelling in Game Engine

Control many parameters variance in the timeline



Sequencer

- **Track:** In order to reference actors in your sequence. Any character, prop, camera, effect, or other actors can be referenced and manipulated in Sequencer
- **Property Track:** Property of reference actors in track
- **Timeline:** A line describing time in discrete frames
- **Key Frame:** The key fames can manipulate properties. Upon reaching a key frame in the timeline, the track's properties are updated to reflect the values you have defined at that point
- **Sequence:** Sequencer's data

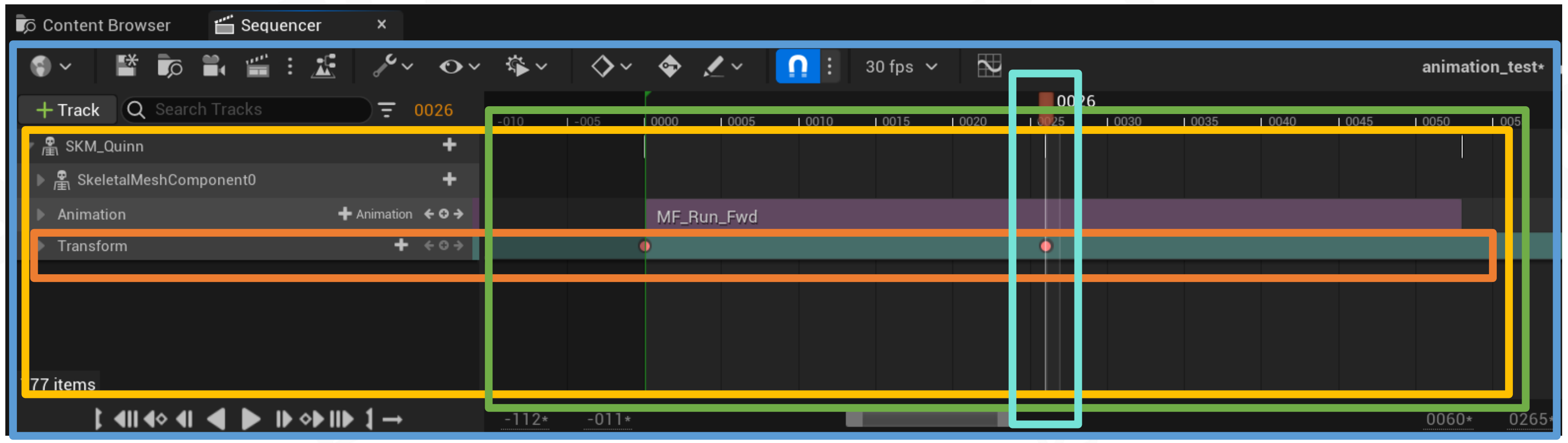
🟨 =Track

🟧 =Property Track

🟩 =Timeline

⬜ =Key Frame

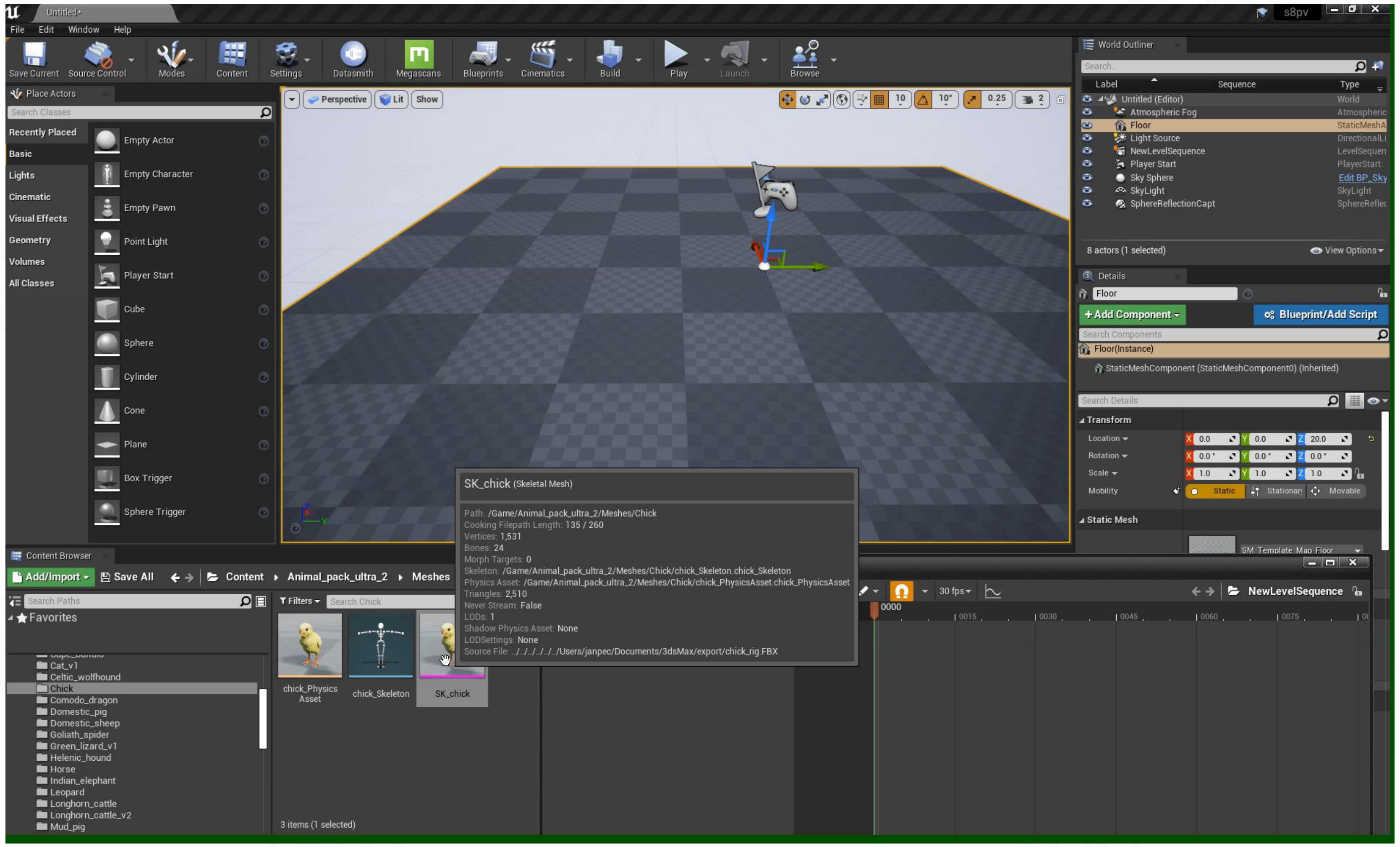
🟦 =Sequence



Sequencer –Bind Objects to Track

How to let the sequencer control my “chick”

- Bind the “chick” to Track



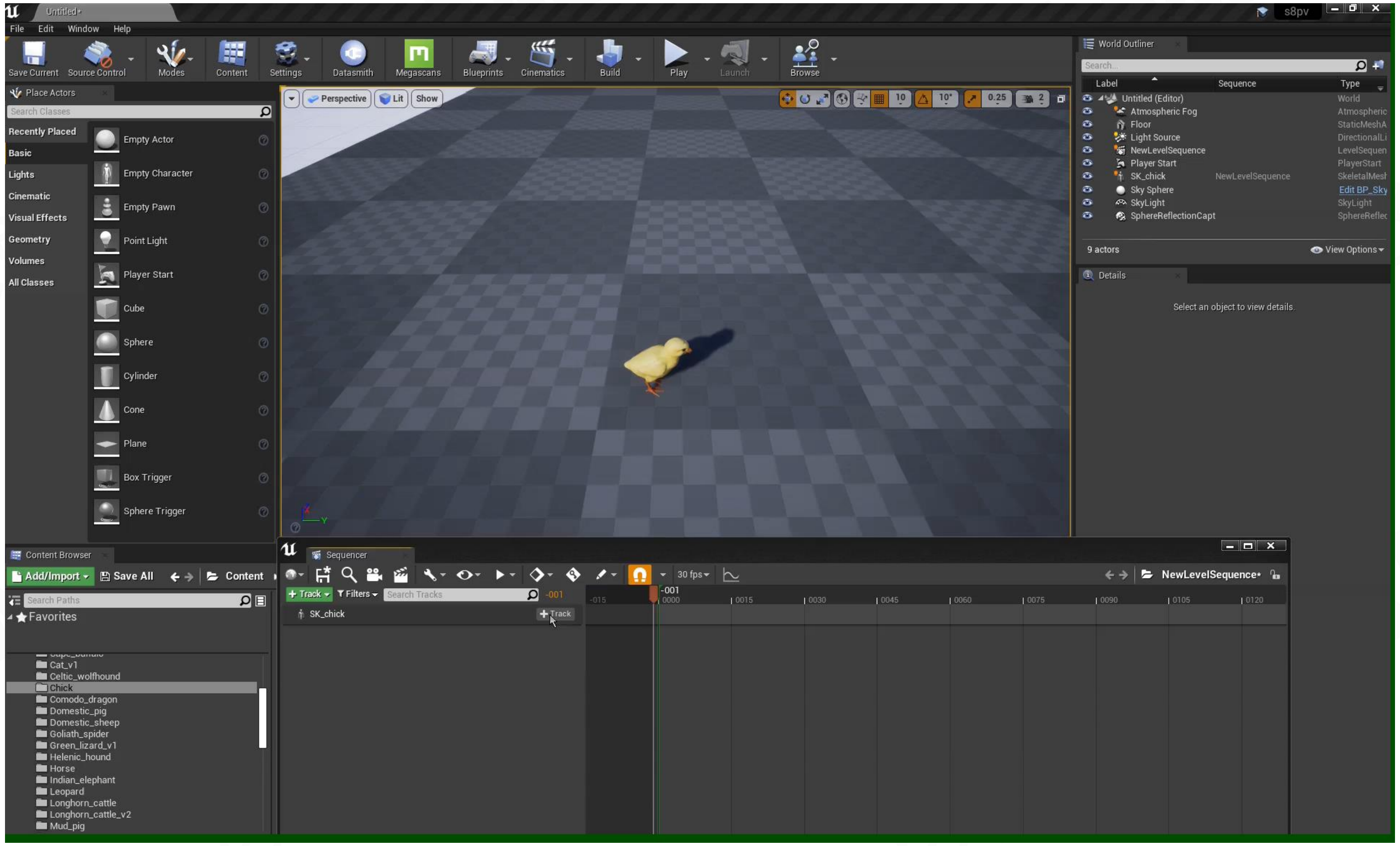
SK_chick (Skeletal Mesh)

Path: /Game/Animal_pack_ultra_2/Meshes/Chick
Cooking Filepath Length: 135 / 260
Vertices: 1,531
Bones: 24
Morph Targets: 0
Skeleton: /Game/Animal_pack_ultra_2/Meshes/Chick/chick_Skeleton.chick_Skeleton
Physics Asset: /Game/Animal_pack_ultra_2/Meshes/Chick/chick_PhysicsAsset.chick_PhysicsAsset
Triangles: 2,510
Never Stream: False
LODs: 1
Shadow Physics Asset: None
LODSettings: None
Source File: ../../../../../../Users/janpec/Documents/3dsMax/export/chick_rig.FBX

Sequencer – Bind Object Property to Property Track

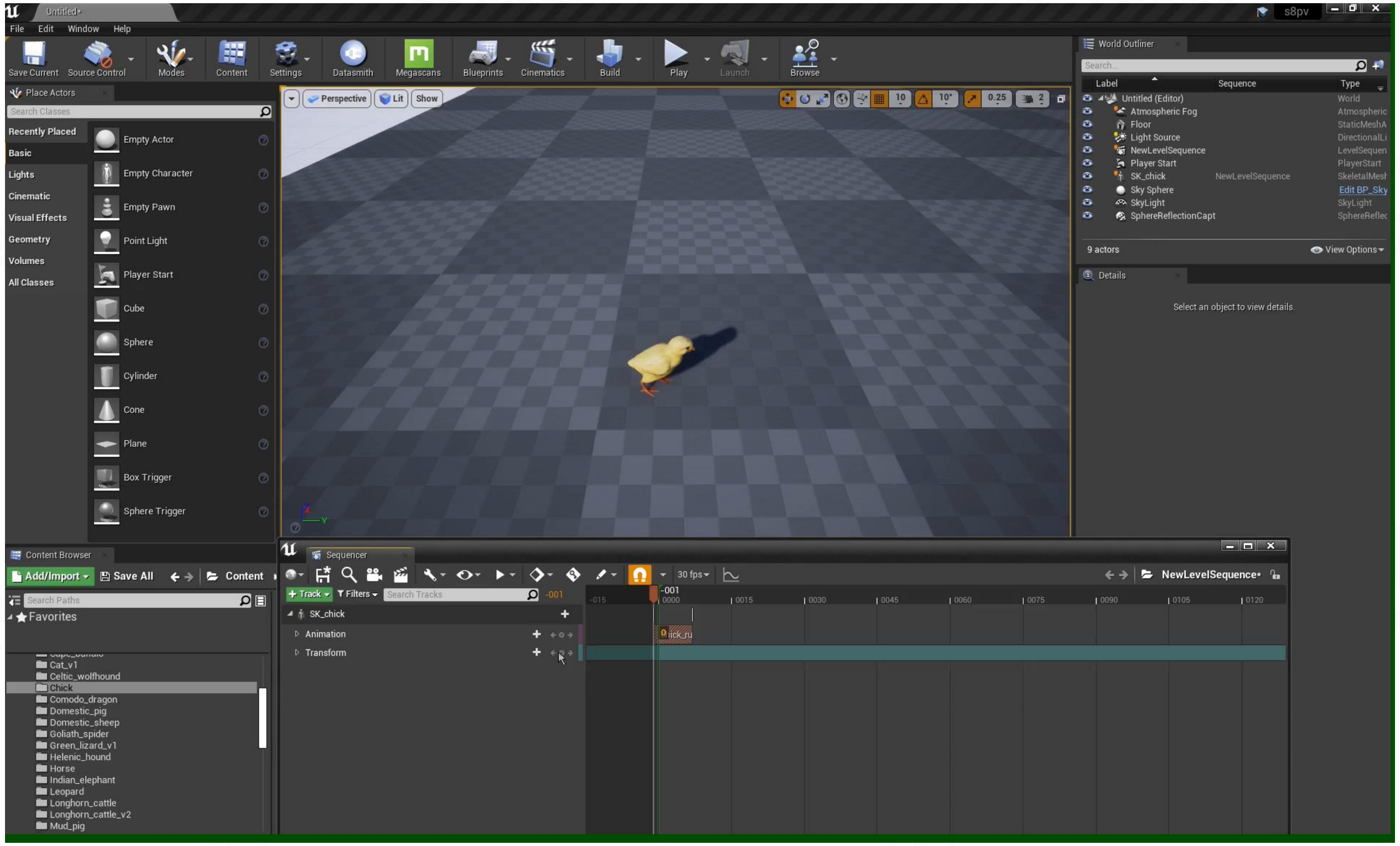
How to control the moving position of the “chick”

- Bind position property to property track



Sequencer – Set Key Frame

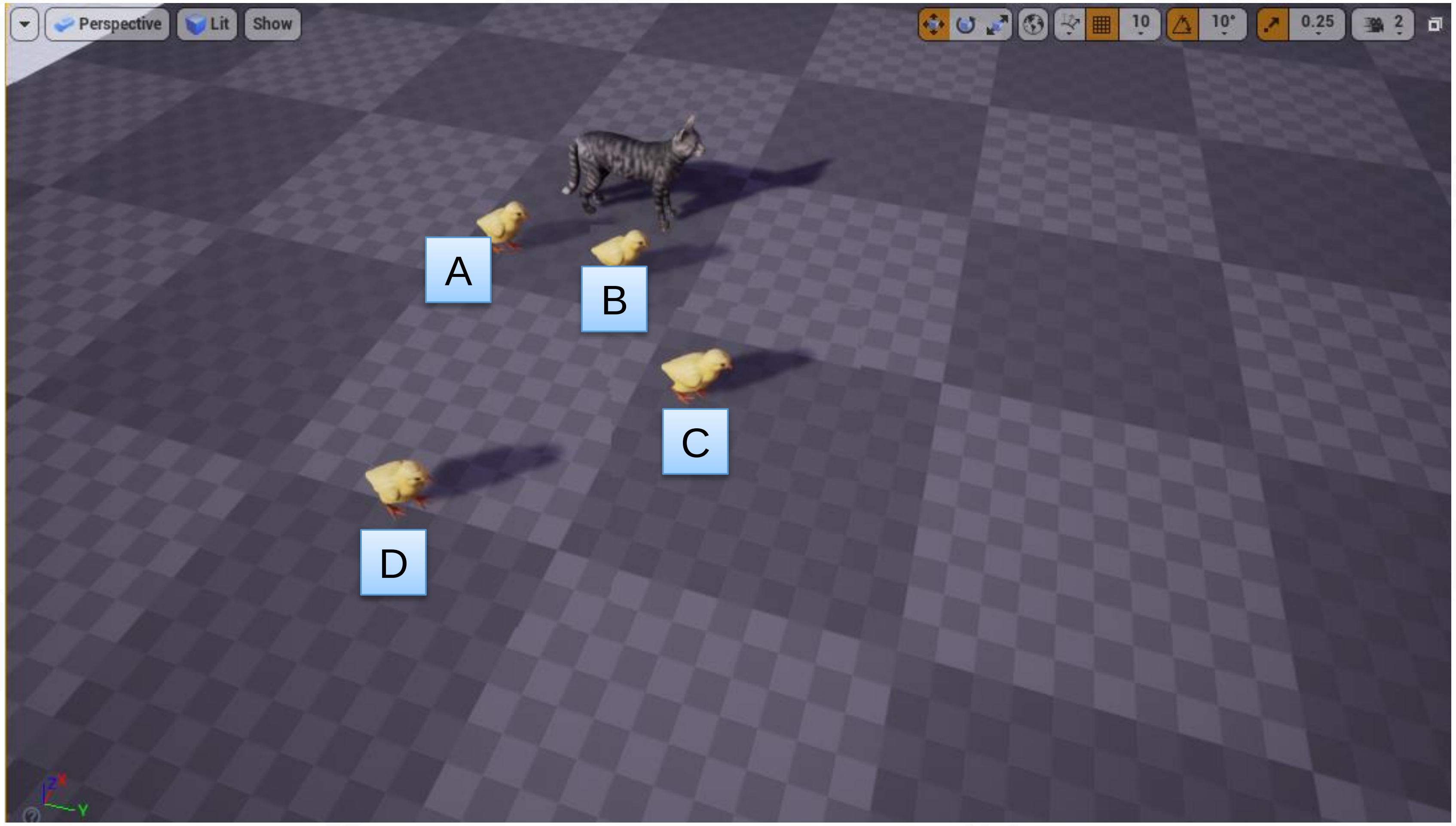
How to make an "chick" reach a specified position



Sequencer – Set Key Frames

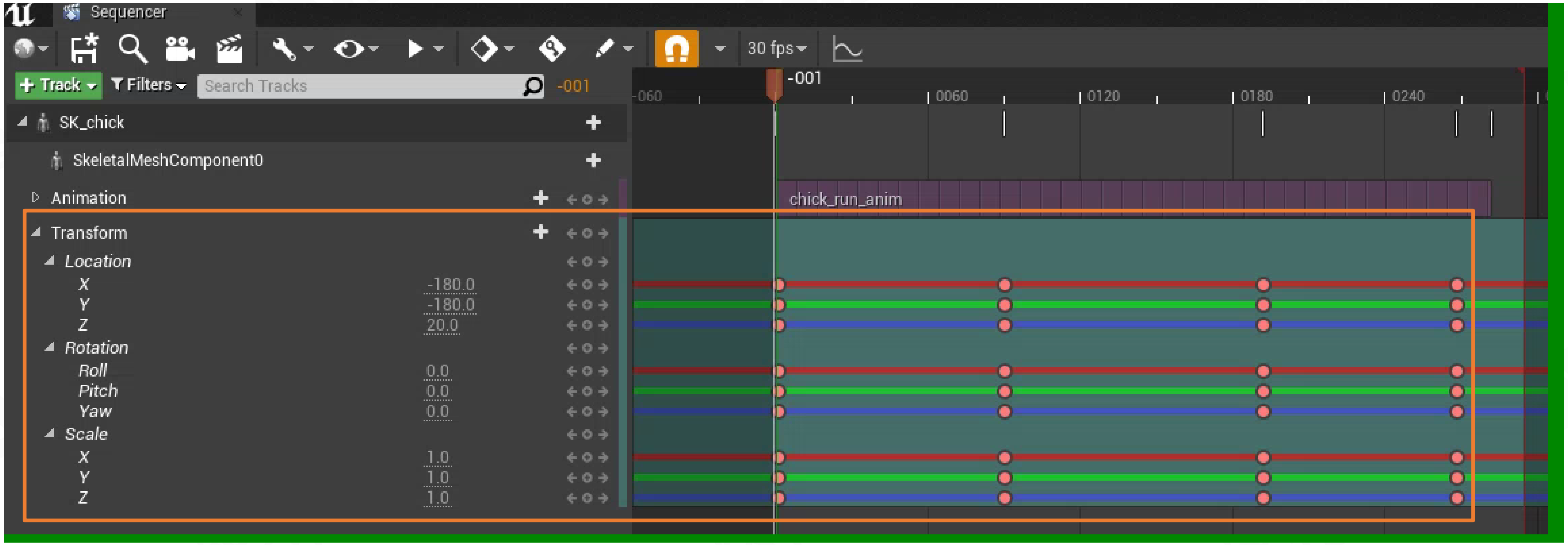
A,B,C,D are key frames.

How “chick” go from A to B to C to D



Sequencer – Interpolate Properties along Key Frames

Similar to animation, set key frames



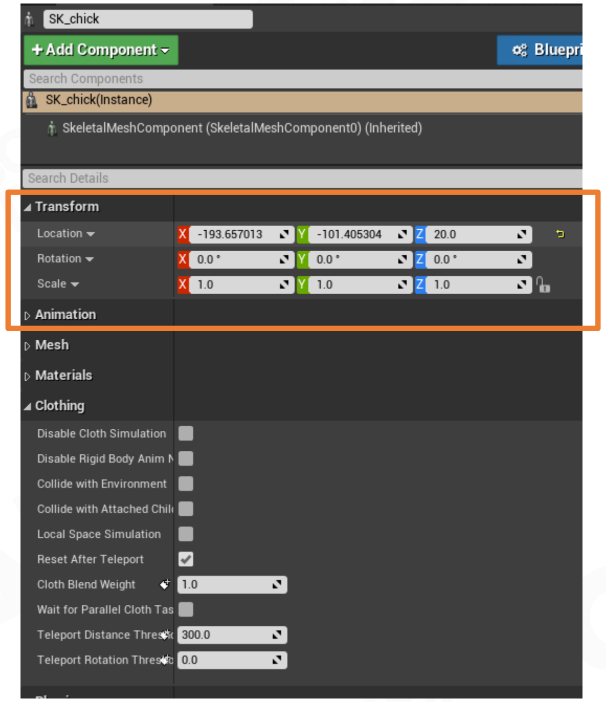
SK_chick
SkeletalMeshComponent0
Animation
Transform
Location
X -180.0
Y -180.0
Z 20.0
Rotation
Roll 0.0
Pitch 0.0
Yaw 0.0
Scale
X 1.0
Y 1.0
Z 1.0

Reflection and Gameplay

Reflection is Foundation of Sequencer

Any data in game engine can be bind into track based on reflection system

“chick” transform



SK_chick

+ Add Component | Bluepri

Search Components

SK_chick(Instance)

SkeletalMeshComponent (SkeletalMeshComponent0) (Inherited)

Search Details

Transform

	X	Y	Z
Location	-193.657013	-101.405304	20.0
Rotation	0.0 °	0.0 °	0.0 °
Scale	1.0	1.0	1.0

Animation

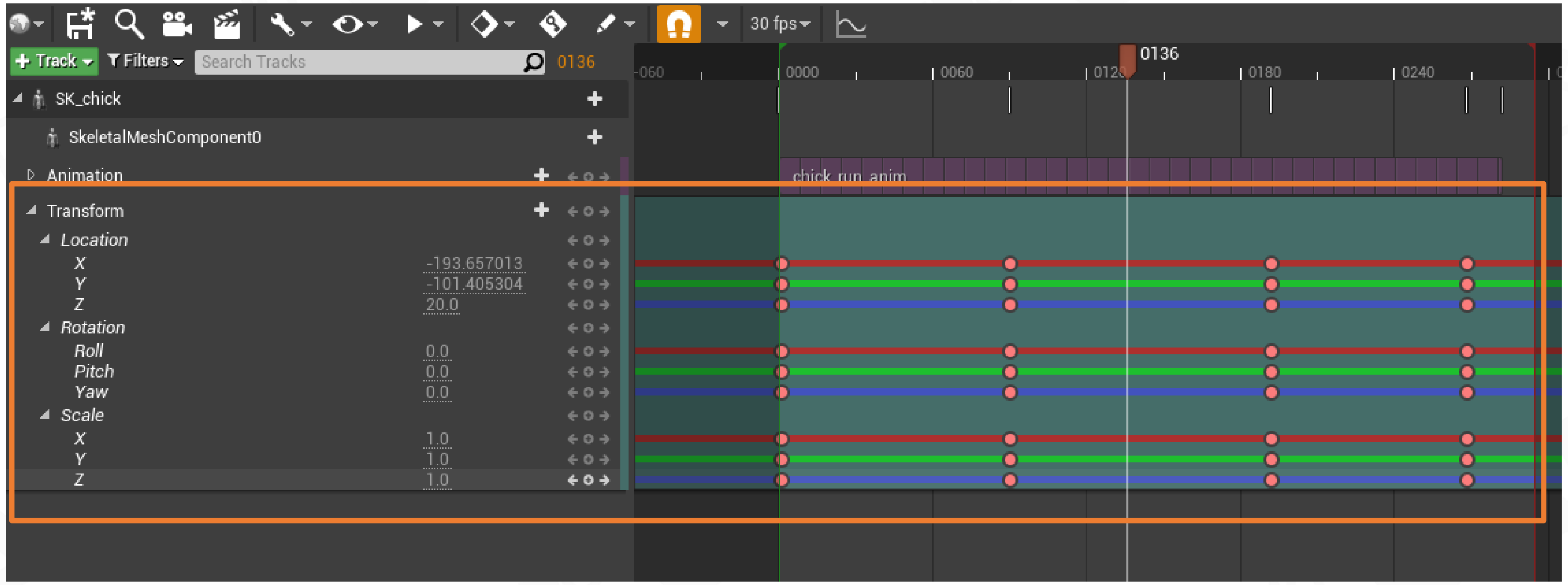
Mesh

Materials

Clothing

Disable Cloth Simulation	☐
Disable Rigid Body Anim N	☐
Collide with Environment	☐
Collide with Attached Chil	☐
Local Space Simulation	☐
Reset After Teleport	✓
Cloth Blend Weight	1.0
Wait for Parallel Cloth Tas	☐
Teleport Distance Thres	300.0
Teleport Rotation Thres	0.0

Sequencer property track



30 fps

+ Track | Filters | Search Tracks | 0136

-060 | 0000 | 0060 | 0120 | 0136 | 0180 | 0240

SK_chick

SkeletalMeshComponent0

Animation — chick_run_anim

Transform

Location

X	-193.657013
Y	-101.405304
Z	20.0

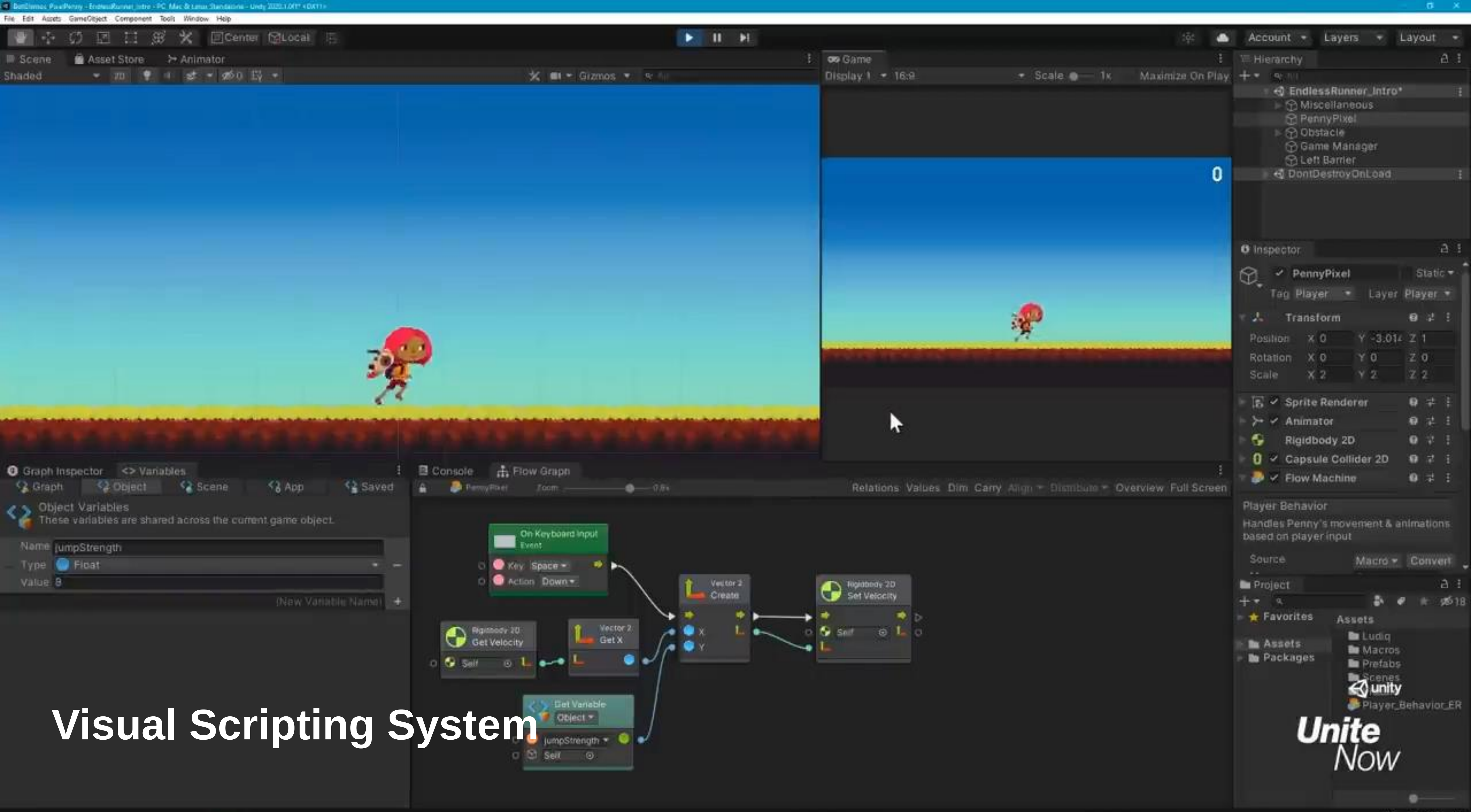
Rotation

Roll	0.0
Pitch	0.0
Yaw	0.0

Scale

X	1.0
Y	1.0
Z	1.0

Complexity of Game Play



Object Variables
These variables are shared across the current game object.

Name jumpStrength
Type Float
Value 8

PennyPixel
Tag Player
Layer Player

Transform
Position X 0 Y -3.014 Z 1
Rotation X 0 Y 0 Z 0
Scale X 2 Y 2 Z 2

Sprite Renderer
Animator
Rigidbody 2D
Capsule Collider 2D
Flow Machine

Player Behavior
Handles Penny's movement & animations based on player input

Source Macro

Key Space
Action Down

Visual Scripting System

Hard Code Method for More Feature (1/2)

```
class Human: public Object{
    void Jump(){
        // do something ...
    }
}
```

```
void CallFunction(Object* instance, string type_name, string func_name)
{
    if(type_name == "Human"){
        Human* human = (Human*)instance;
        if(func_name == "Jump"){
            human->Jump();
        }
    }
}
```

Hard Code Method for More Feature (2/2)

```
class Human: public Object{
    void Jump(){
        // do something ...
    }

    void StopJump(){
        // do something ...
    }
}
```

```
void CallFunction(Object* instance, string type_name, string func_name)
{
    if(type_name == "Human"){
        Human* human = (Human*)instance;
        if(func_name == "Jump"){
            human->Jump();
        }
        else if(func_name == "StopJump"){
            human->StopJump();
        }
    }
}
```

A Common Solution - Reflection

In computer science, reflective programming or reflection is the ability of a process to examine, introspect, and modify its own structure and behavior.

Java Reflection

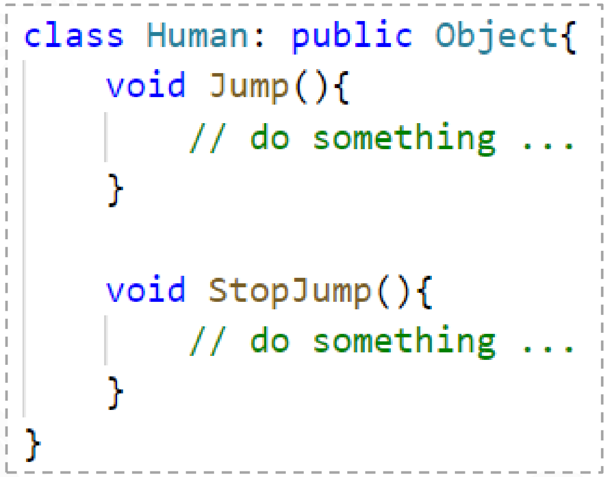
```
package Demo;
public class Test {
    public int m_filed;
    public void print()
    {
        System.out.print("call print().");
    };
}
```

```
package Demo;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
public class Demo {
    public static void main(String[] args) throws Exception{
        Class<?> cls = Class.forName("Demo.Test");
        Object obj = cls.getConstructor().newInstance();
        Field filed_accessor = cls.getField("m_filed");
        filed_accessor.set(obj, 2);

        Method method_accessor = cls.getMethod("print");
        method_accessor.invoke(obj);
    }
}
```

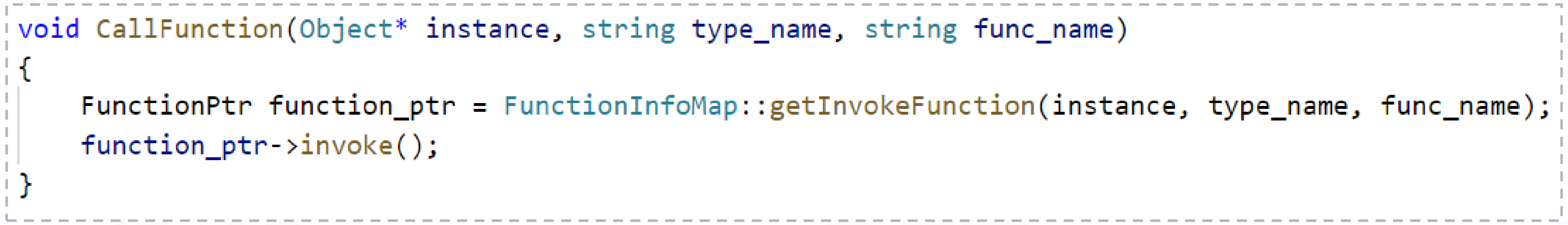
Reflection Build the Bridge between Code and Tools

Using reflection to generate a code meta information map

- class_name, func_name and para_name
- generate accessor and invoker



```cpp
class Human: public Object{
    void Jump(){
        // do something ...
    }

    void StopJump(){
        // do something ...
    }
}
```



```cpp
void CallFunction(Object* instance, string type_name, string func_name)
{
    FunctionPtr function_ptr = FunctionInfoMap::getInvokeFunction(instance, type_name, func_name);
    function_ptr->invoke();
}
```

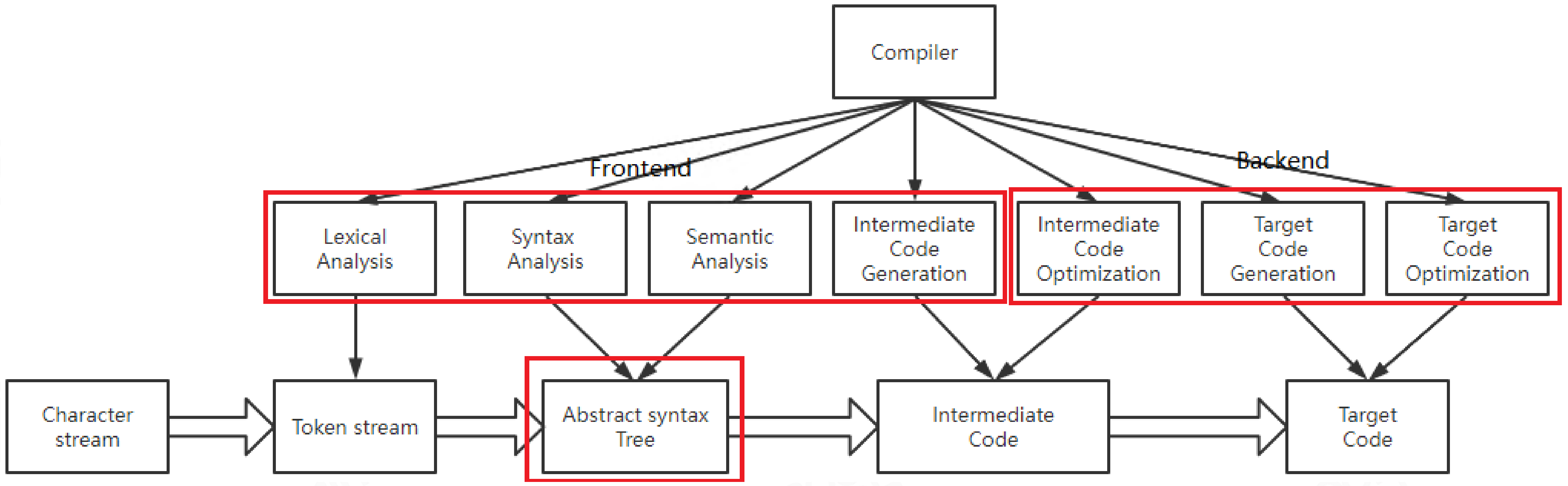
Once and for all!

How to Implement Reflection in C++

- Collect type info from code
- Generate code to provide accessors for fields and methods
- Manage all accessors with a <string,accessor> map

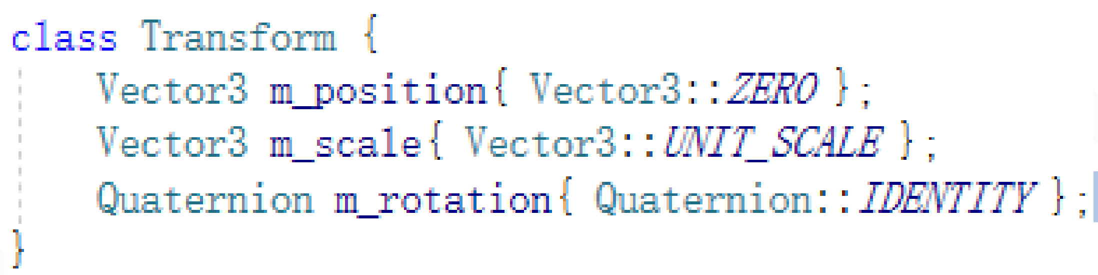
How to Get Type Info from Code(1/2)

General Programming Language(GPL) Compilation Process

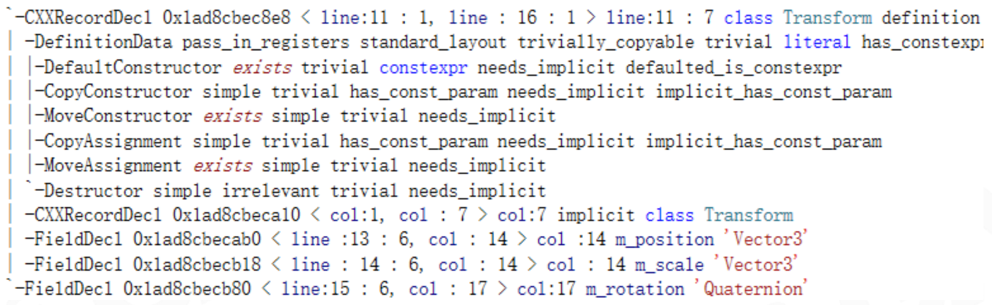


How to Get Type Info from Code(2/2)

Abstract Syntax Tree(AST): An abstract representation of the syntax structure of source code. It represents the syntax structure of programming language in the form of a tree, and each node in the tree represents a construct in the source code.



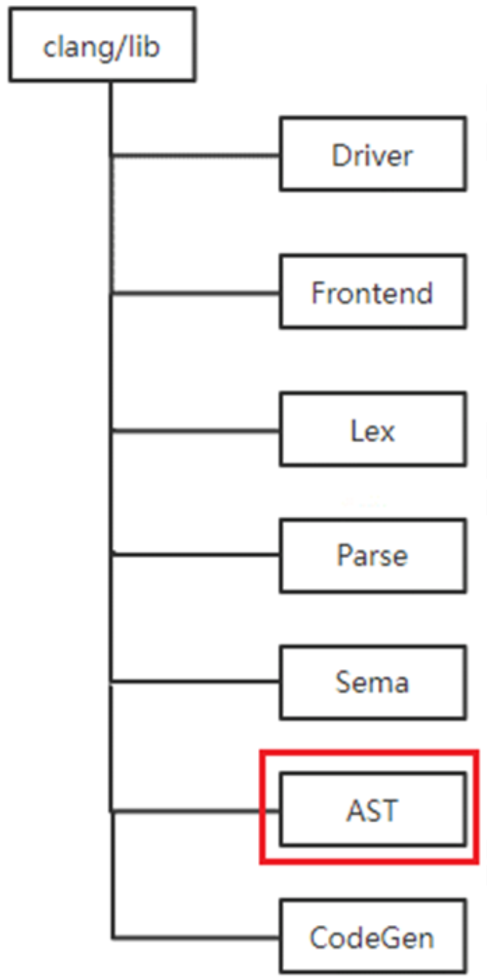
```
class Transform {
    Vector3 m_position{ Vector3::ZERO };
    Vector3 m_scale{ Vector3::UNIT_SCALE };
    Quaternion m_rotation{ Quaternion::IDENTITY };
}
```



```
`-CXXRecordDecl 0x1ad8cbec8e8 < line:11 : 1, line : 16 : 1 > line:11 : 7 class Transform definition
 | -DefinitionData pass_in_registers standard_layout trivially_copyable trivial literal has_constexp
 | |-DefaultConstructor exists trivial constexpr needs_implicit defaulted_is_constexpr
 | |-CopyConstructor simple trivial has_const_param needs_implicit implicit_has_const_param
 | |-MoveConstructor exists simple trivial needs_implicit
 | |-CopyAssignment simple trivial has_const_param needs_implicit implicit_has_const_param
 | |-MoveAssignment exists simple trivial needs_implicit
 | `-Destructor simple irrelevant trivial needs_implicit
 | -CXXRecordDecl 0x1ad8cbeca10 < col:1, col : 7 > col:7 implicit class Transform
 | -FieldDecl 0x1ad8cbecab0 < line :13 : 6, col : 14 > col :14 m_position 'Vector3'
 | -FieldDecl 0x1ad8cbecb18 < line : 14 : 6, col : 14 > col : 14 m_scale 'Vector3'
`-FieldDecl 0x1ad8cbecb80 < line:15 : 6, col : 17 > col:17 m_rotation 'Quaternion'
```

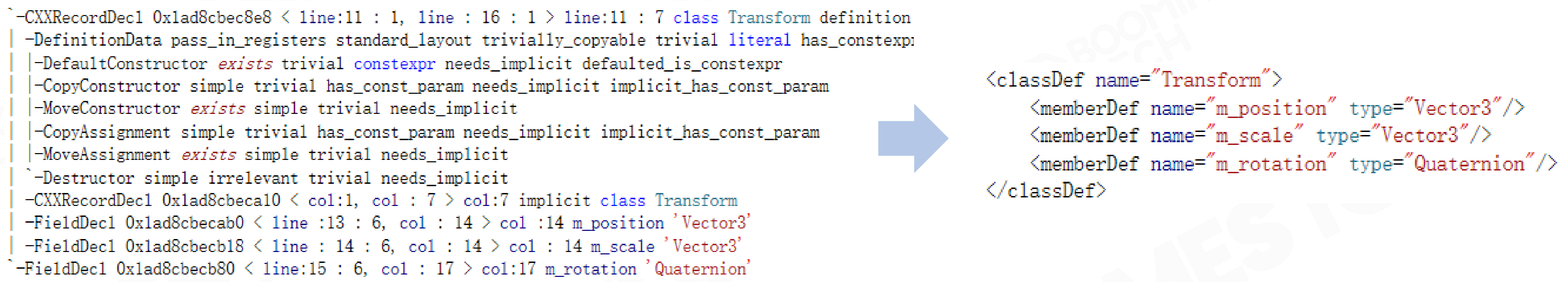
Why Piccolo Use Clang

One of Clang's main goals is to provide a library-based architecture, so that the compiler could interoperate with other tools that interact with source code.



Generate Schema From AST

- Parsing AST, such as type name, field name, field type, etc.
- Build a temporary schema of data in memory

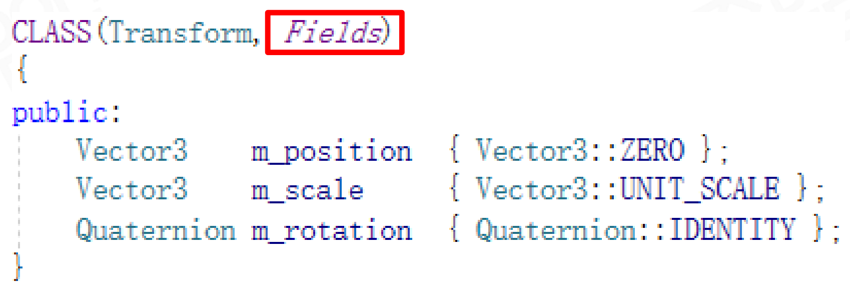


```
`-CXXRecordDecl 0x1ad8cbec8e8 < line:11 : 1, line : 16 : 1 > line:11 : 7 class Transform definition
 | -DefinitionData pass_in_registers standard_layout trivially_copyable trivial literal has_constexp
 | |-DefaultConstructor exists trivial constexpr needs_implicit defaulted_is_constexpr
 | |-CopyConstructor simple trivial has_const_param needs_implicit implicit_has_const_param
 | |-MoveConstructor exists simple trivial needs_implicit
 | |-CopyAssignment simple trivial has_const_param needs_implicit implicit_has_const_param
 | |-MoveAssignment exists simple trivial needs_implicit
 | `-Destructor simple irrelevant trivial needs_implicit
 | -CXXRecordDecl 0x1ad8cbeca10 < col:1, col : 7 > col:7 implicit class Transform
 | -FieldDecl 0x1ad8cbecab0 < line :13 : 6, col : 14 > col :14 m_position 'Vector3'
 | -FieldDecl 0x1ad8cbecb18 < line : 14 : 6, col : 14 > col : 14 m_scale 'Vector3'
`-FieldDecl 0x1ad8cbecb80 < line:15 : 6, col : 17 > col:17 m_rotation 'Quaternion'
```

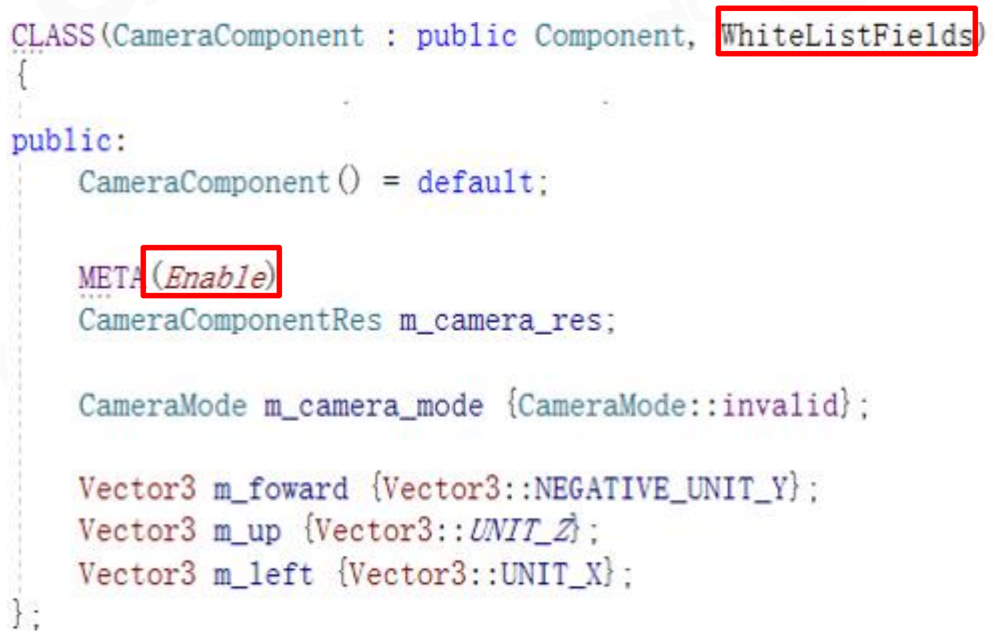
```
<classDef name="Transform">
    <memberDef name="m_position" type="Vector3"/>
    <memberDef name="m_scale" type="Vector3"/>
    <memberDef name="m_rotation" type="Quaternion"/>
</classDef>
```

Precise Control of Reflection Scope

In the actual scenario, we need to add a lot of tag information to identify the purpose of the type.



```
CLASS(Transform, Fields)
{
public:
    Vector3    m_position  { Vector3::ZERO };
    Vector3    m_scale     { Vector3::UNIT_SCALE };
    Quaternion m_rotation  { Quaternion::IDENTITY };
}
```

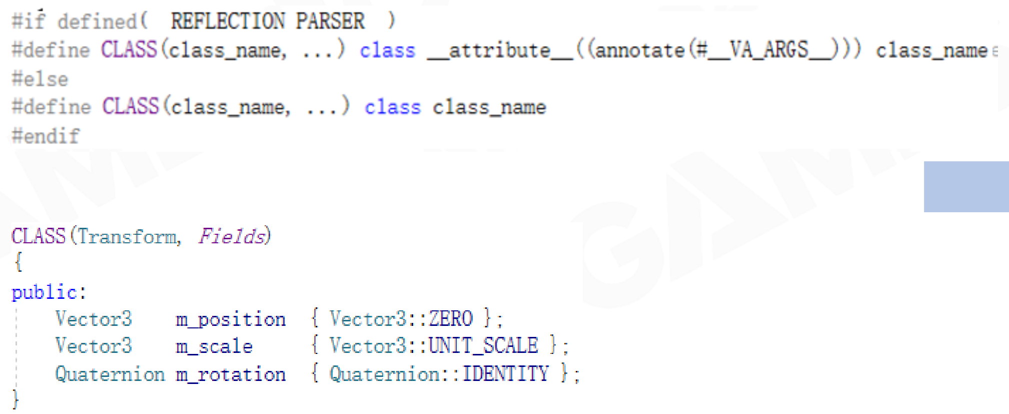


```
CLASS(CameraComponent : public Component, WhiteListFields)
{

public:
    CameraComponent() = default;

    META(Enable)
    CameraComponentRes m_camera_res;

    CameraMode m_camera_mode {CameraMode::invalid};

    Vector3 m_foward {Vector3::NEGATIVE_UNIT_Y};
    Vector3 m_up {Vector3::UNIT_Z};
    Vector3 m_left {Vector3::UNIT_X};
};
```

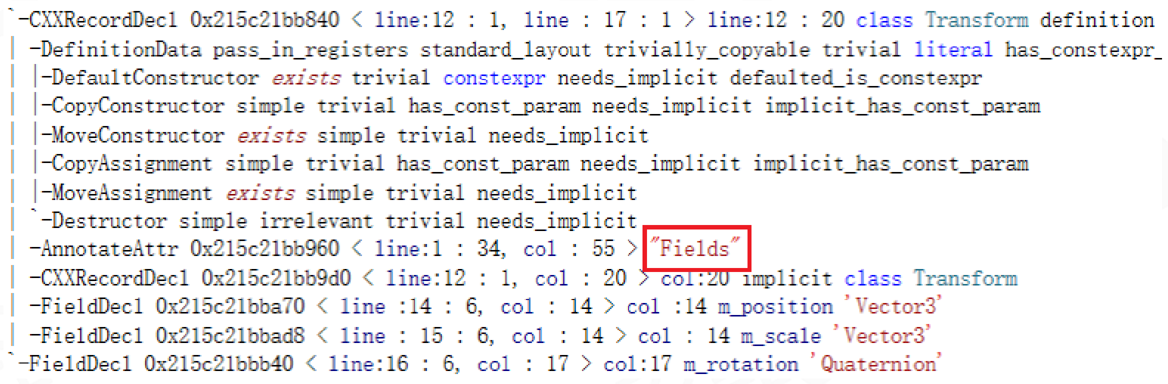
Use Marco to Add Reflection Controls

Add tags by *__attribute__*

- __attribute__ is a source code annotation provided by clang. In the code, the required data types can be captured by using these macros.
- Define a “CLASS” macro to distinguish between precompile and compile.
  - When precompiling, define “_REFOECTION_PARSER_” macro in *meta_parser* to make the attribute information effective



```
#if defined(  REFLECTION PARSER  )
#define CLASS(class_name, ...) class __attribute__((annotate(#__VA_ARGS__))) class_name
#else
#define CLASS(class_name, ...) class class_name
#endif


CLASS(Transform, Fields)
{
public:
    Vector3    m_position  { Vector3::ZERO };
    Vector3    m_scale     { Vector3::UNIT_SCALE };
    Quaternion m_rotation  { Quaternion::IDENTITY };
}
```

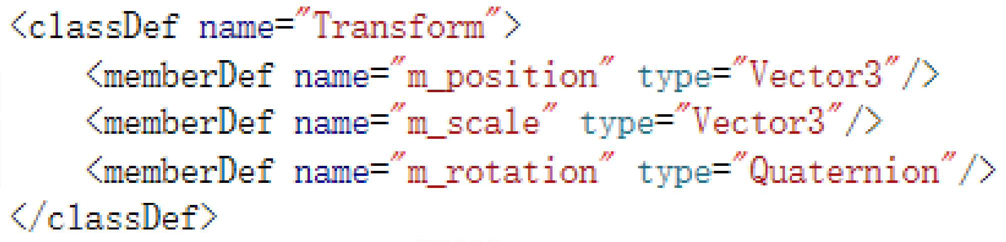


```
`-CXXRecordDecl 0x215c21bb840 < line:12 : 1, line : 17 : 1 > line:12 : 20 class Transform definition
 | -DefinitionData pass_in_registers standard_layout trivially_copyable trivial literal has_constexpr_
 | |-DefaultConstructor exists trivial constexpr needs_implicit defaulted_is_constexpr
 | |-CopyConstructor simple trivial has_const_param needs_implicit implicit_has_const_param
 | |-MoveConstructor exists simple trivial needs_implicit
 | |-CopyAssignment simple trivial has_const_param needs_implicit implicit_has_const_param
 | |-MoveAssignment exists simple trivial needs_implicit
 | `-Destructor simple irrelevant trivial needs_implicit
 | -AnnotateAttr 0x215c21bb960 < line:1 : 34, col : 55 > "Fields"
 | -CXXRecordDecl 0x215c21bb9d0 < line:12 : 1, col : 20 > col:20 implicit class Transform
 | -FieldDecl 0x215c21bba70 < line :14 : 6, col : 14 > col :14 m_position 'Vector3'
 | -FieldDecl 0x215c21bbad8 < line : 15 : 6, col : 14 > col : 14 m_scale 'Vector3'
`-FieldDecl 0x215c21bbb40 < line:16 : 6, col : 17 > col:17 m_rotation 'Quaternion'
```

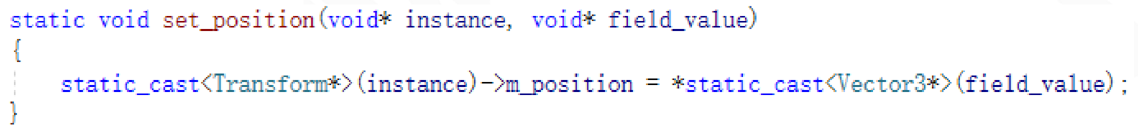
Reflection Accessors

Generate reflection accessors using schemas

- For classes, we need to generate type info getters
- For fields, we need to generate setters and getters that can access them
- For functions, we need to generate invoker that can invoke tem



```xml
<classDef name="Transform">
    <memberDef name="m_position" type="Vector3"/>
    <memberDef name="m_scale" type="Vector3"/>
    <memberDef name="m_rotation" type="Quaternion"/>
</classDef>
```



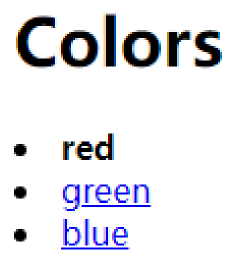
```cpp
static void set_position(void* instance, void* field_value)
{
    static_cast<Transform*>(instance)->m_position = *static_cast<Vector3*>(field_value);
}
```

Code Rendering (1/2)

The same type of business code structure is the same

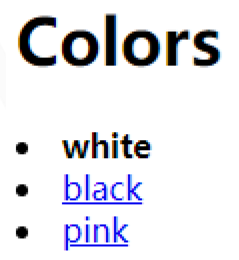
```
<h1>Colors</h1>
<li><strong>red</strong></li>
<li><a href="#Green">green</a></li>
<li><a href="#Blue">blue</a></li>
```

```
<h1>Colors</h1>
<li><strong>white</strong></li>
<li><a href="#Black">black</a></li>
<li><a href="#Pink">pink</a></li>
```



Colors

- **red**
- green
- blue



Colors

- **white**
- black
- pink

```
<h1>{{header}}</h1>

{{#items}}
  {{#first}}
    <li><strong>{{name}}</strong></li>
  {{/first}}
  {{#link}}
    <li><a href="{{url}}">{{name}}</a></li>
  {{/link}}
{{/items}}

{{#empty}}
  <p>The list is empty.</p>
{{/empty}}
```

...

Code Rendering (2/2)

Code Rendering is the process of collecting data (if any) and loading related templates (or sending output directly). The collected data is then applied to the associated template. The final output is sent to the user.

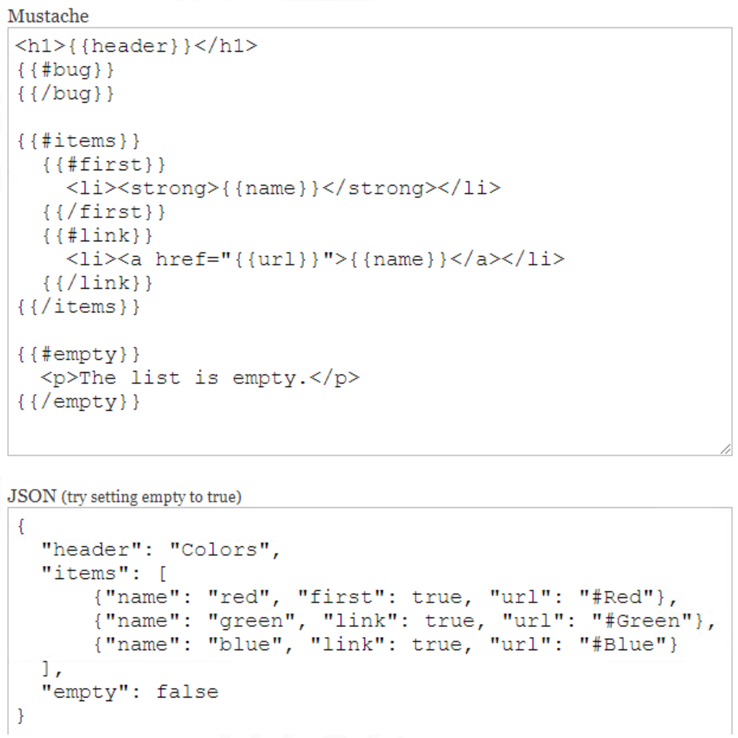
Pros:

- Strong separation of code and data

Code Rendering – Mustache

Mustache is a web template system.

It is named "Mustache" because of heavy use of braces, {{ }}, that resemble a sideways moustache.

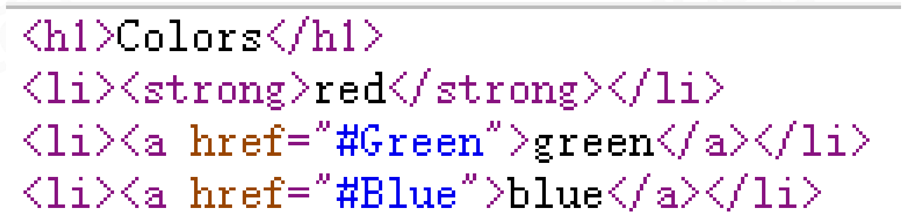


Mustache

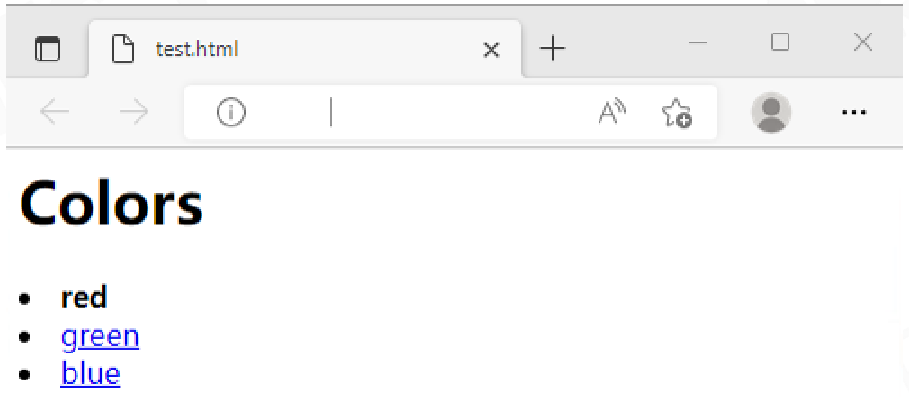
```
<h1>{{header}}</h1>
{{#bug}}
{{/bug}}

{{#items}}
  {{#first}}
    <li><strong>{{name}}</strong></li>
  {{/first}}
  {{#link}}
    <li><a href="{{url}}">{{name}}</a></li>
  {{/link}}
{{/items}}

{{#empty}}
  <p>The list is empty.</p>
{{/empty}}
```

JSON (try setting empty to true)

```
{
  "header": "Colors",
  "items": [
      {"name": "red", "first": true, "url": "#Red"},
      {"name": "green", "link": true, "url": "#Green"},
      {"name": "blue", "link": true, "url": "#Blue"}
  ],
  "empty": false
}
```



```
<h1>Colors</h1>
<li><strong>red</strong></li>
<li><a href="#Green">green</a></li>
<li><a href="#Blue">blue</a></li>
```

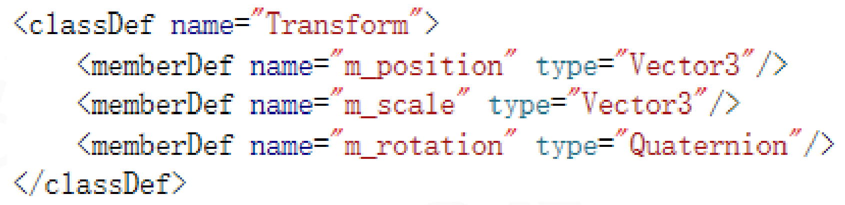


Colors

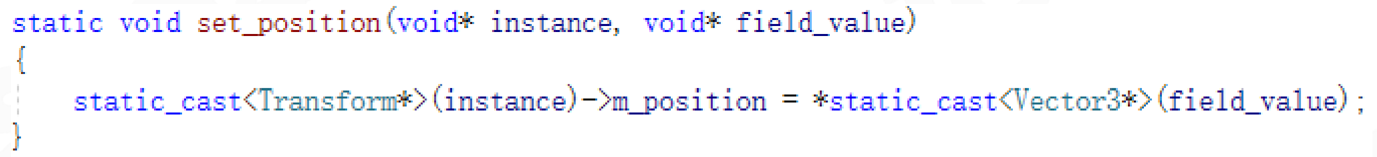
- **red**
- green
- blue

Use Mustache to Code Generation

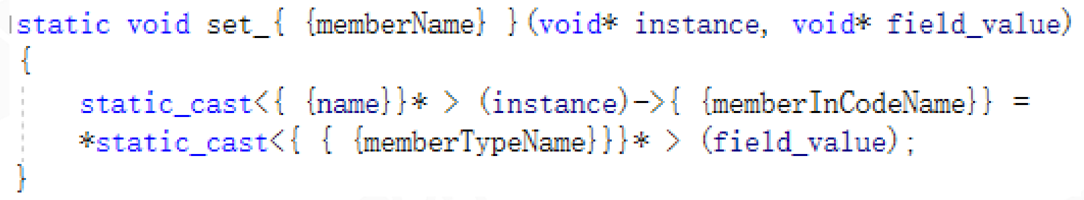
- Implementing business logic using mustache templates
- Generate code through mustache rendering



```
<classDef name="Transform">
    <memberDef name="m_position" type="Vector3"/>
    <memberDef name="m_scale" type="Vector3"/>
    <memberDef name="m_rotation" type="Quaternion"/>
</classDef>
```



```
static void set_position(void* instance, void* field_value)
{
    static_cast<Transform*>(instance)->m_position = *static_cast<Vector3*>(field_value);
}
```

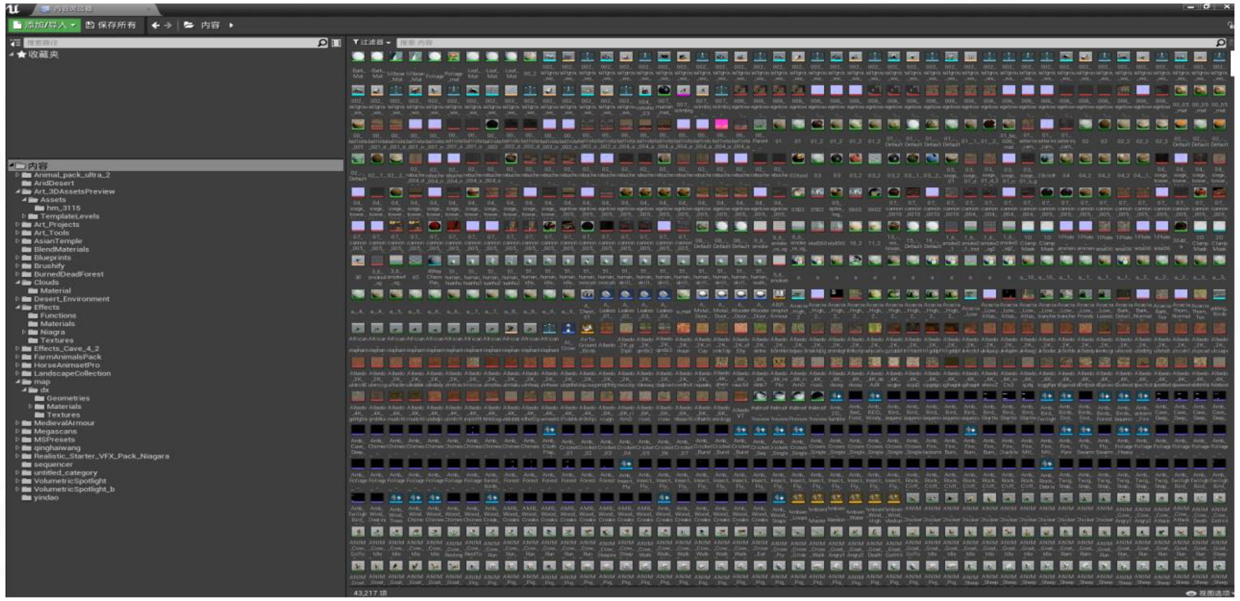


```
static void set_{ {memberName} }(void* instance, void* field_value)
{
    static_cast<{ {name}}* > (instance)->{ {memberInCodeName}} =
    *static_cast<{ { {memberTypeName}}}* > (field_value);
}
```

Collaborative Editing

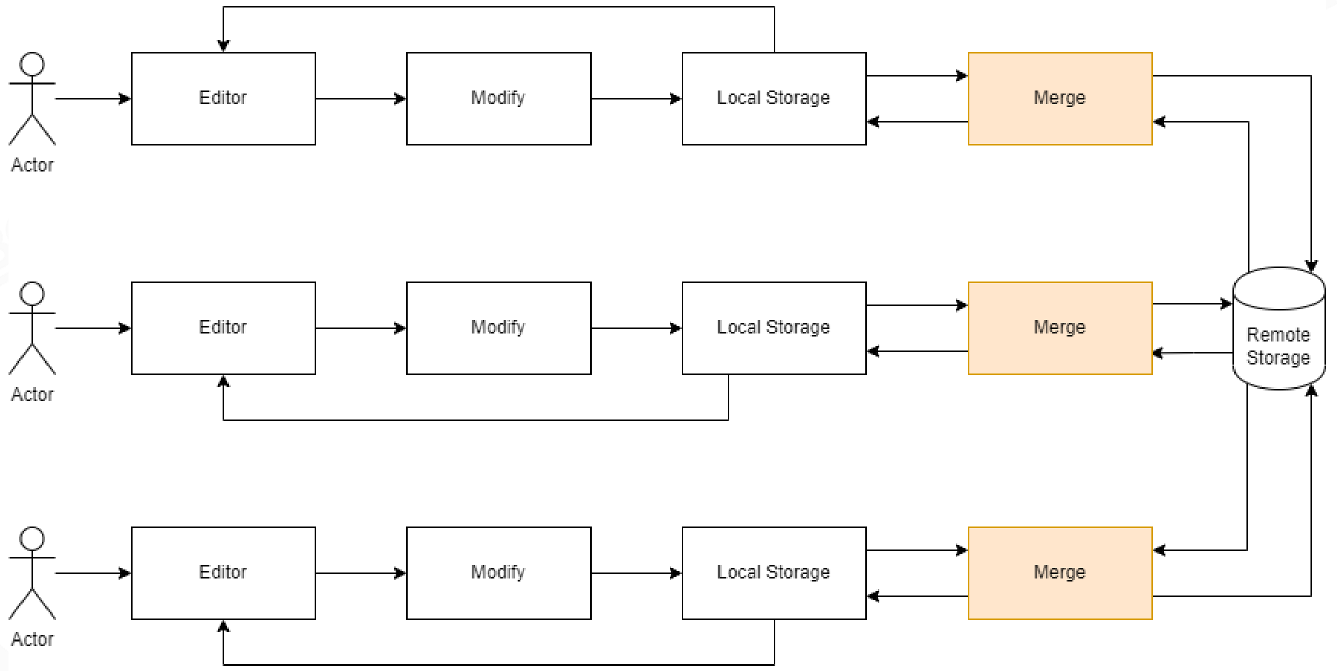
Bottlenecks in Large Projects

- Lots of people work with lots of assets
- Assets version management is very difficult



Merging Conflicts is The Biggest Problem

- Everyone needs to spend a lot of time on merging conflicts when updating or uploading assets



Work Flow

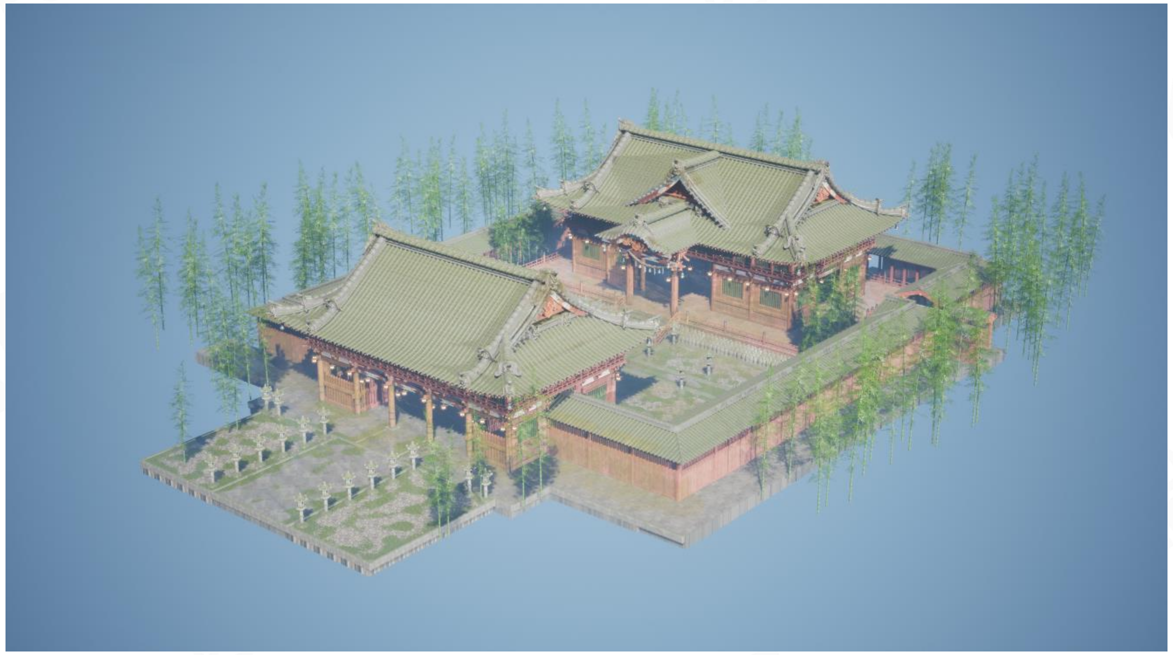
How to Reduce Conflicts

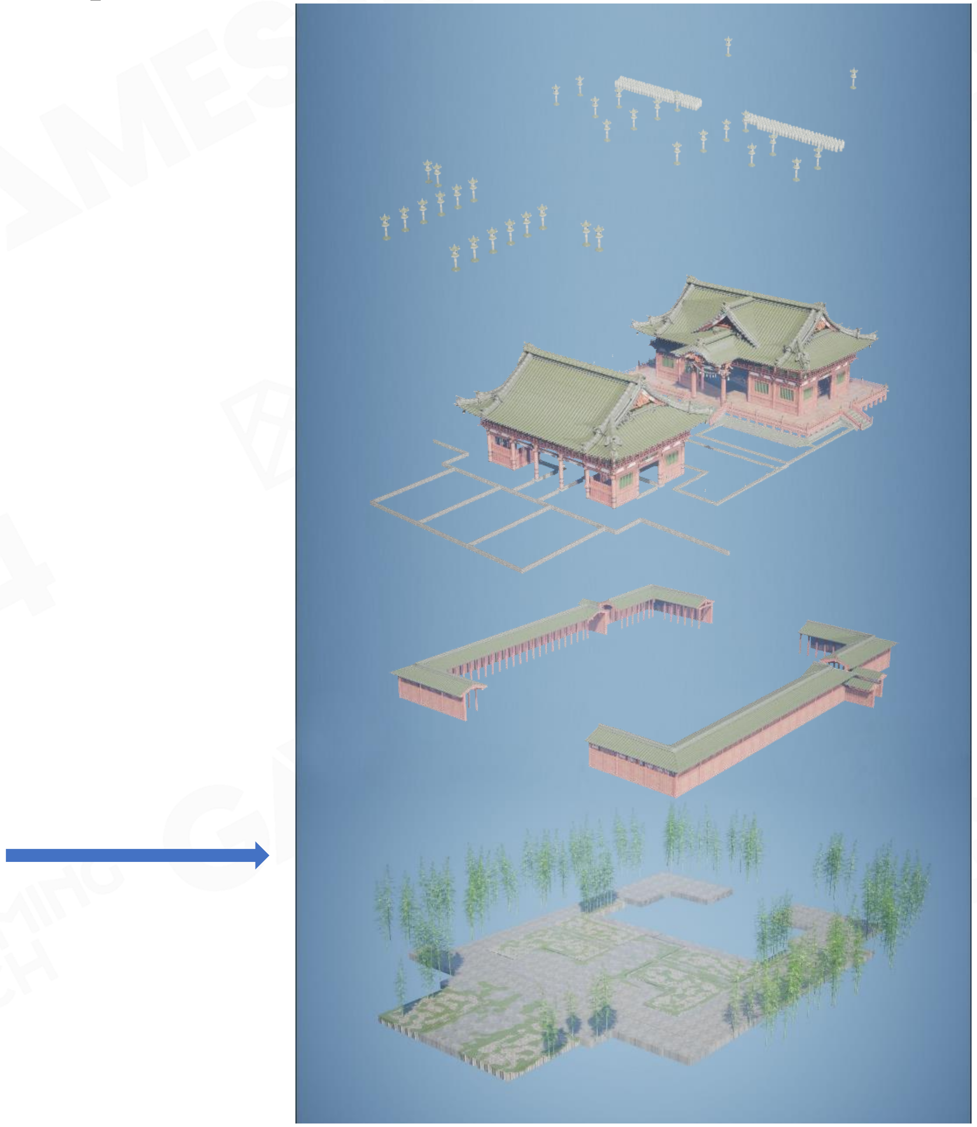
- Split assets into smaller parts to reduce the probability of conflicts
  - Layering the world
  - Divide the world
  - One file per actor(OFPA)

- All people work in the same scene to completely eliminate the conflict

Split Assets – Layering the World (1/2)

- Split the world into many layers, each of which is stored in an asset file
- Different people work at different levels





Split Assets – Layering the World (2/2)

Pros

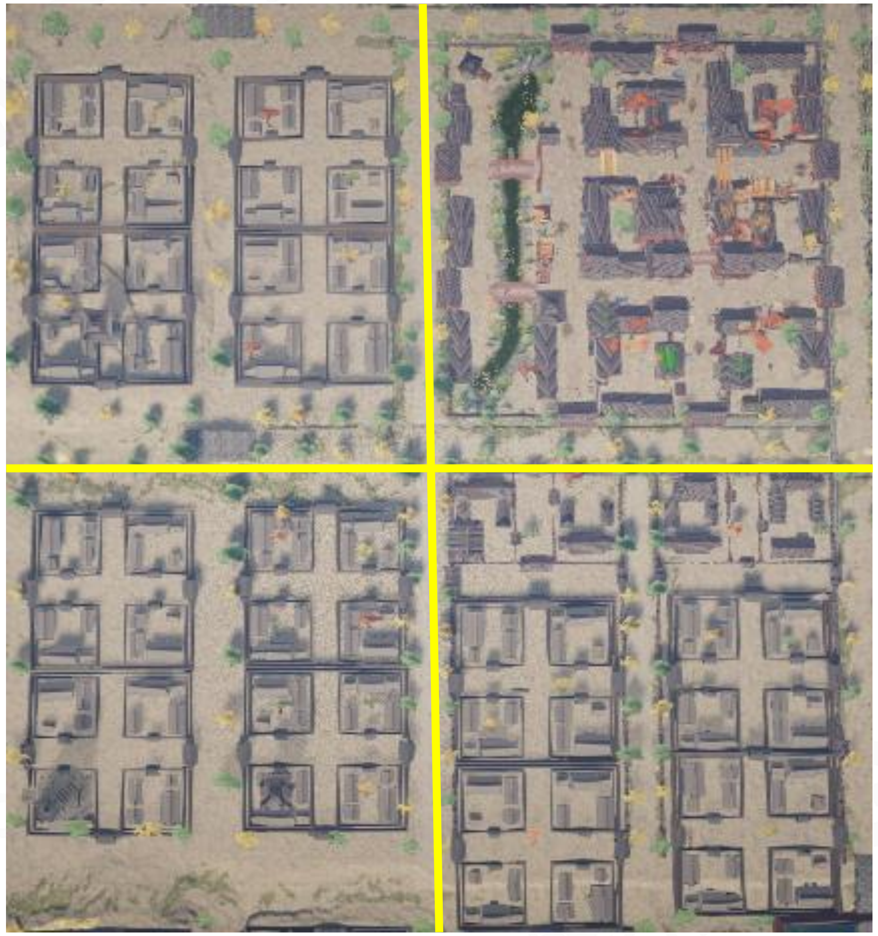
- Appropriate layers would decrease edit confliction
- Layer-based logic available

Cons

- Layer logic may dependents on another layer
- Difficult to reasonably split layers when the world is very complex

Split Assets– Divide the World (1/2)

- The world is divided into fixed size blocks, and each block is saved in an asset file
- Different people work at different blocks



Split Assets – Divide the World (2/2)

Pros

- Location based splitting makes it easy to dynamically expand the world
- Space separating is more intuitive to operator

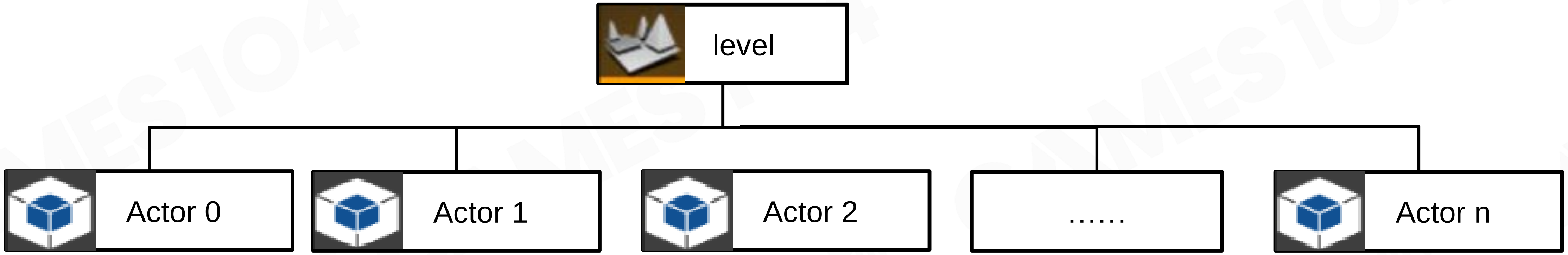
Cons

- Difficult to deal with objects across multiple blocks

One File Per Actor

A splitting method proposed by unreal5

- reduces overlap between users by saving data for instances of Actors in external files, removing the need to save the main Level file when making changes to its Actors
- All Actors are embedded in their respective Level files when cooked



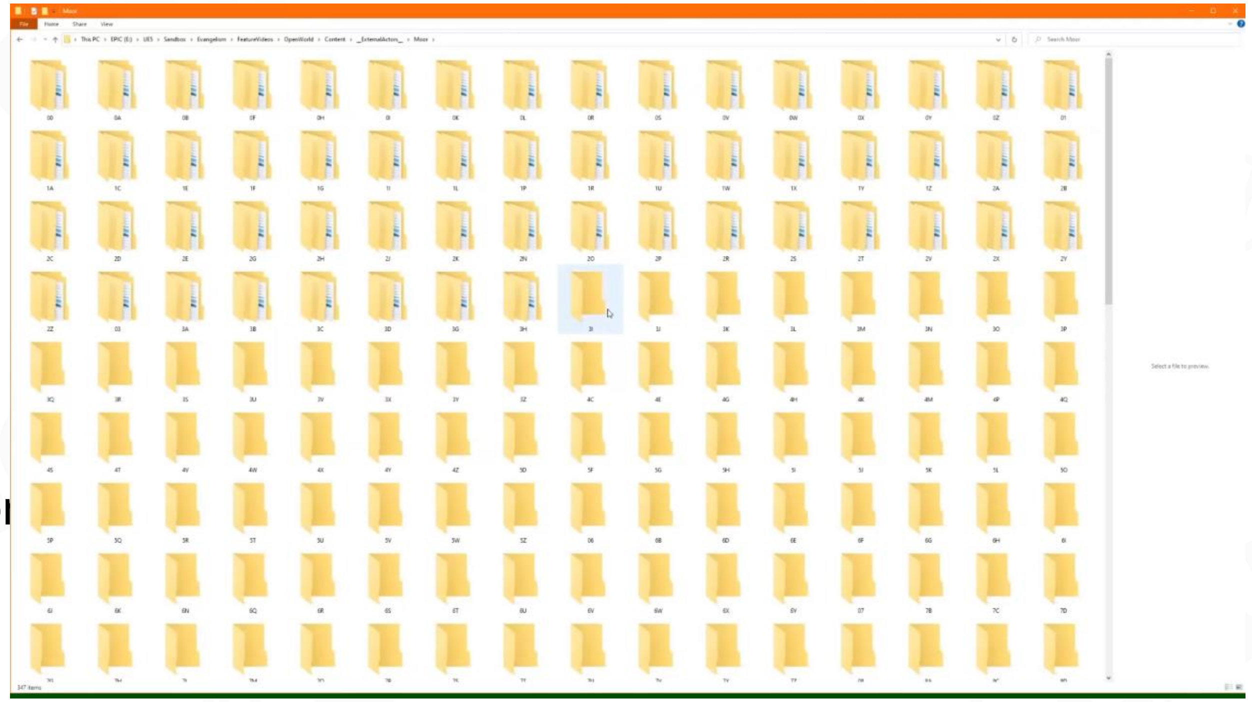
A Special Way to Split Assets – OFPA

Pros

- Fine-grained scene division, fewer edit confliction
- Only need to save objects modified

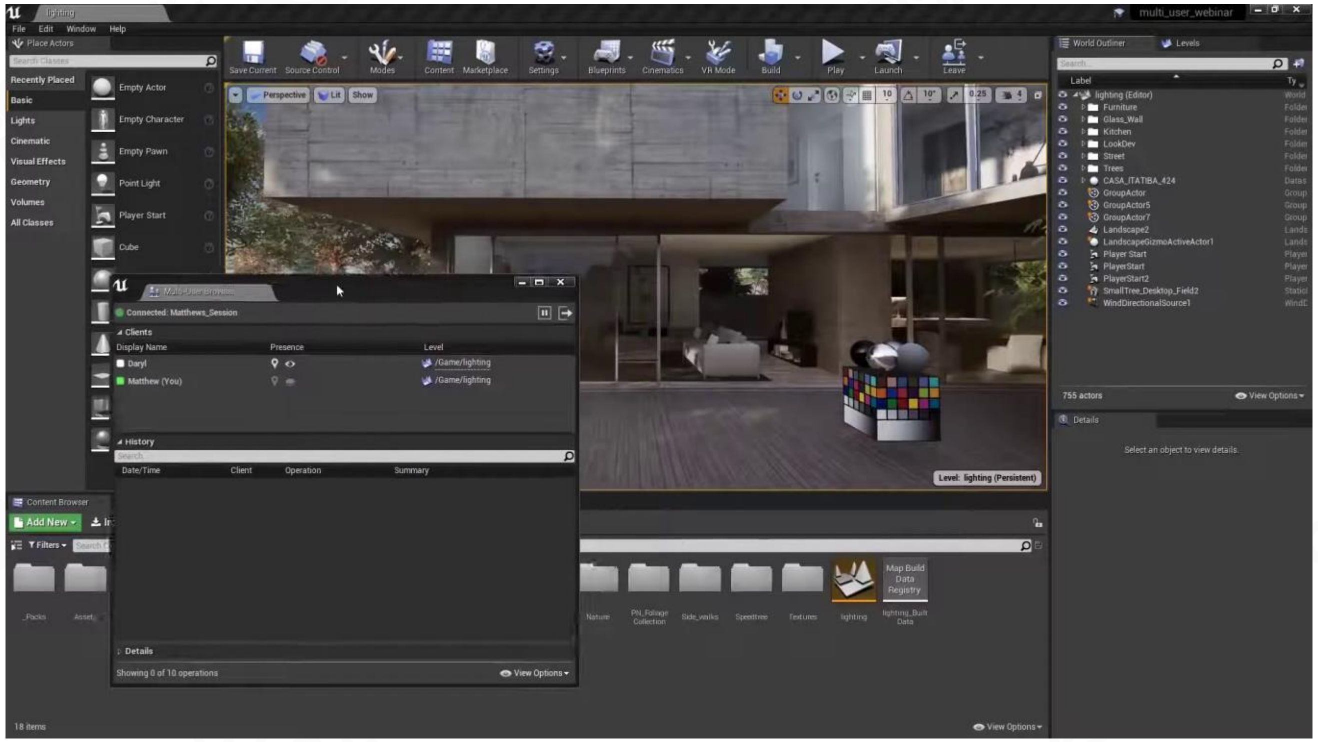
Cons

- Massive files to manage, more burden for versio control
- Cook will be slow down while embedding many OFPA files to level file



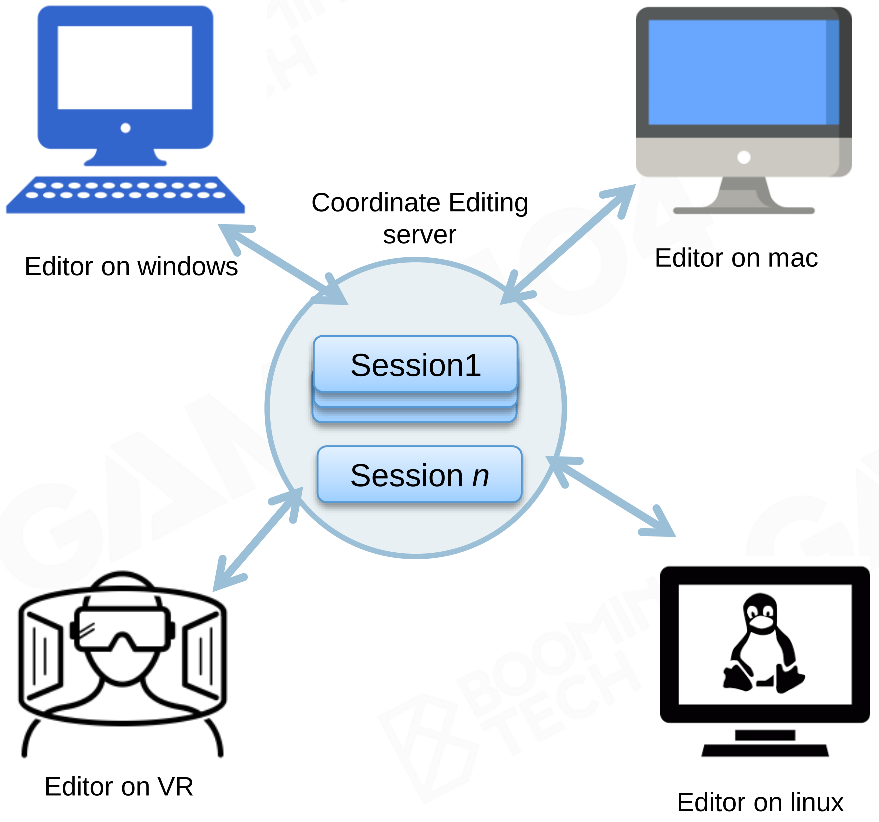
Coordinate Editing in One Scene

Connect multiple instances of world editor together to work collaboratively in a shared editing session, building a single virtual world together with your teammates and colleagues in real time



Display Name	Presence	Level
Daryl		/Game/lighting
Matthew (You)		/Game/lighting

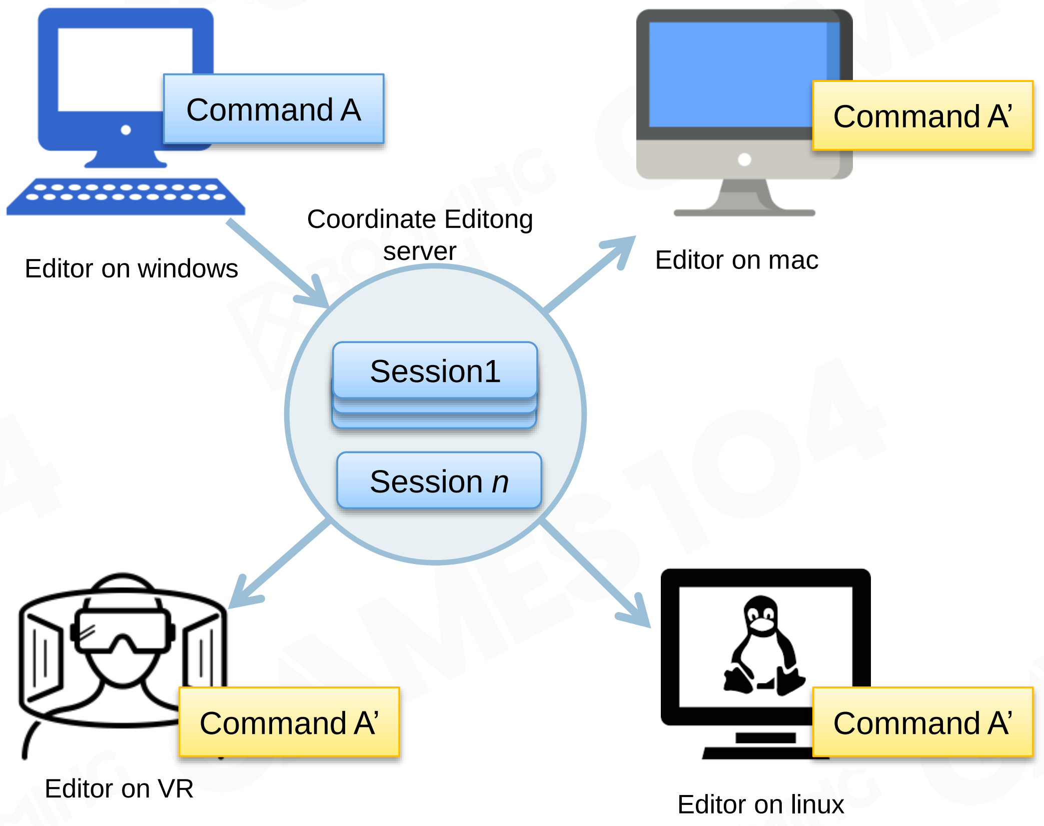
Date/Time	Client	Operation	Summary



How to Synchronize My Operations with Others

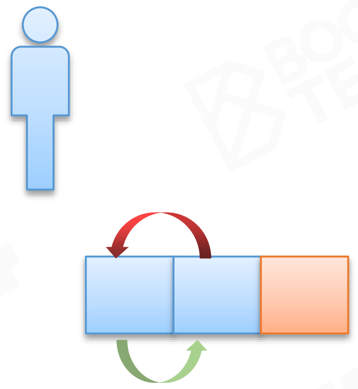
Do you remember command system?

- Serialize my commands and send them to server
- Receive commands from server and deserialize them
- Invoke commands

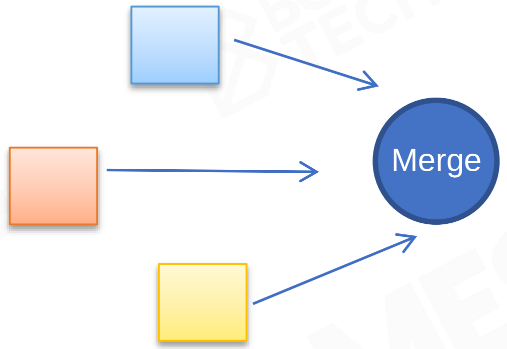


There is A Very Big Challenge

How to ensure the consistency of distributed operations?



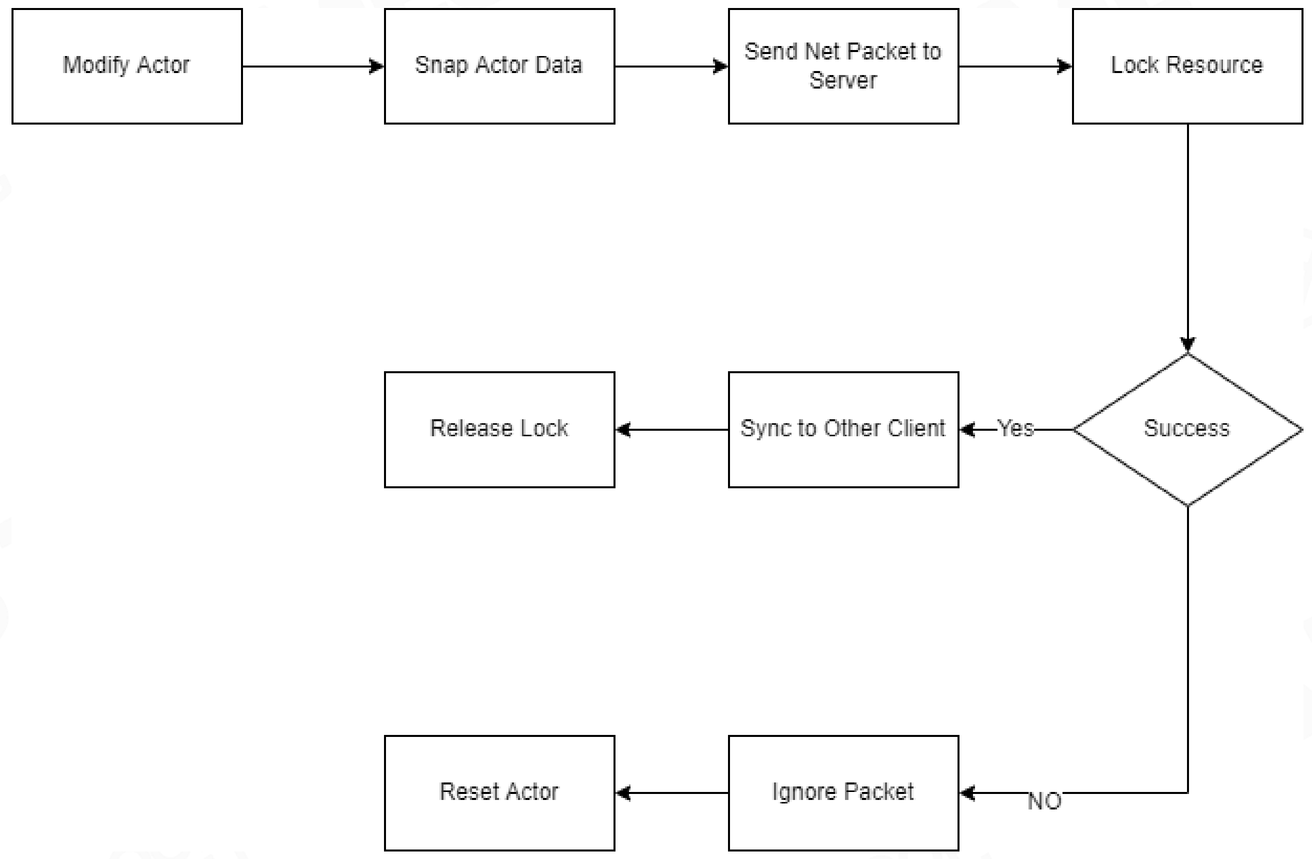
Undo/Redo



Operation Merge

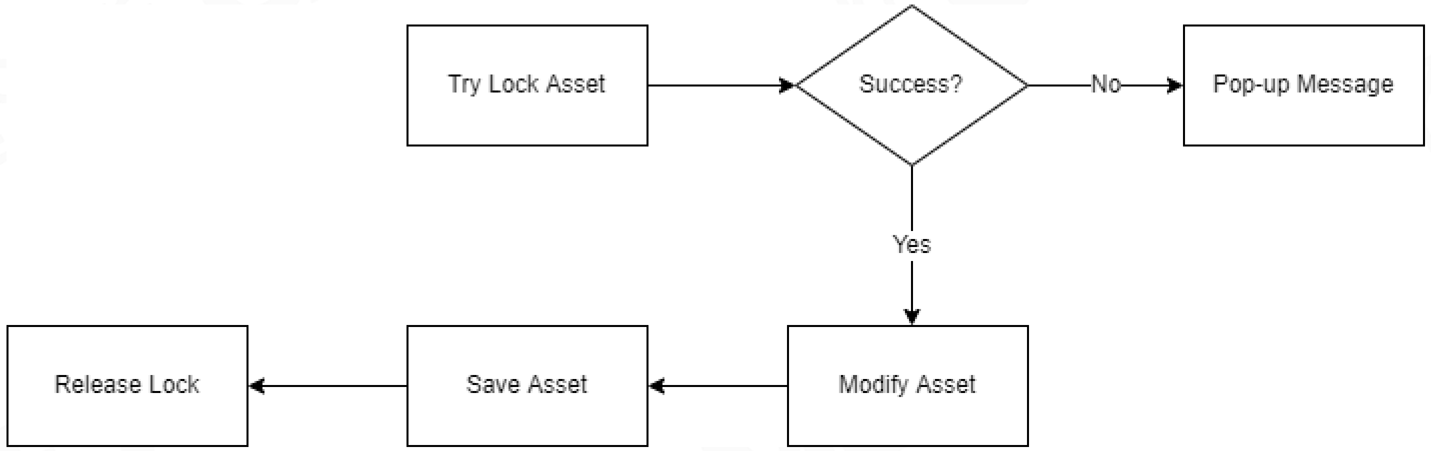
Two Users Cannot Edit The Same Instance at The Same Time

Instance lock: Avoid multiple people modifying the same instance at the same time



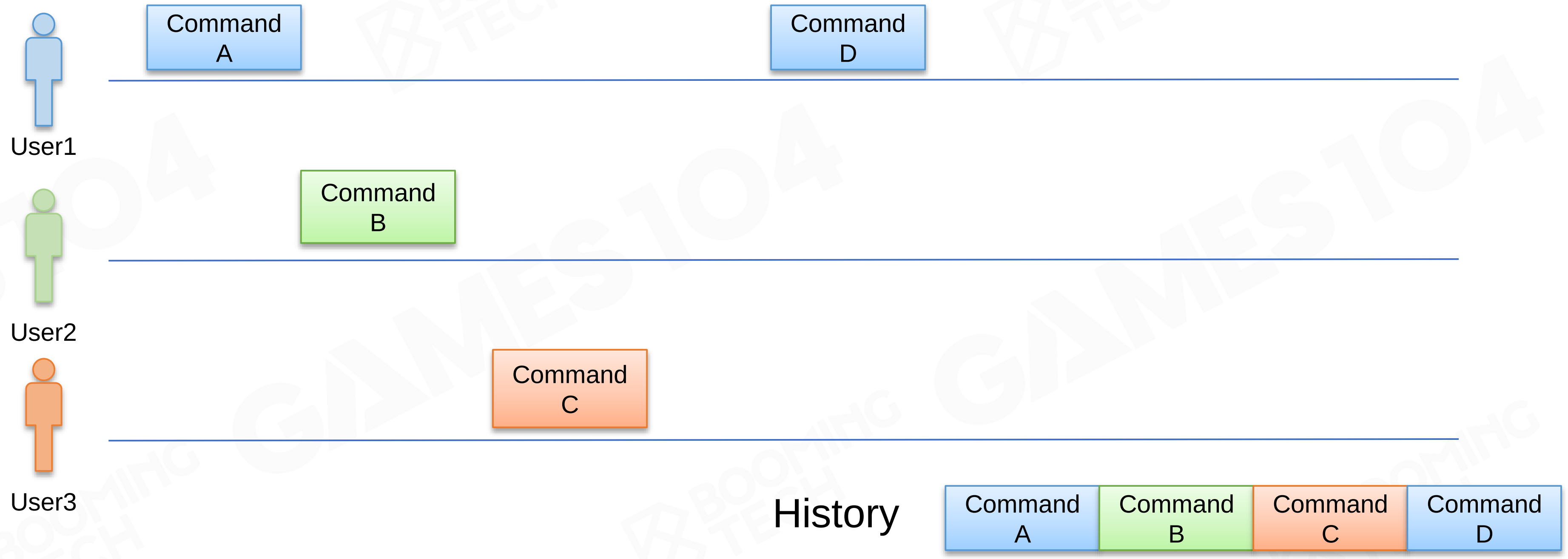
Two Users Cannot Edit The Same Asset at The Same Time

Asset lock: Avoid multiple people modifying the same asset at the same time



But Lock is not Omnipotent (1/2)

If there are three users working in the same world, and now User2 presses the undo button, what do we expect to happen? If he presses the redo button next?



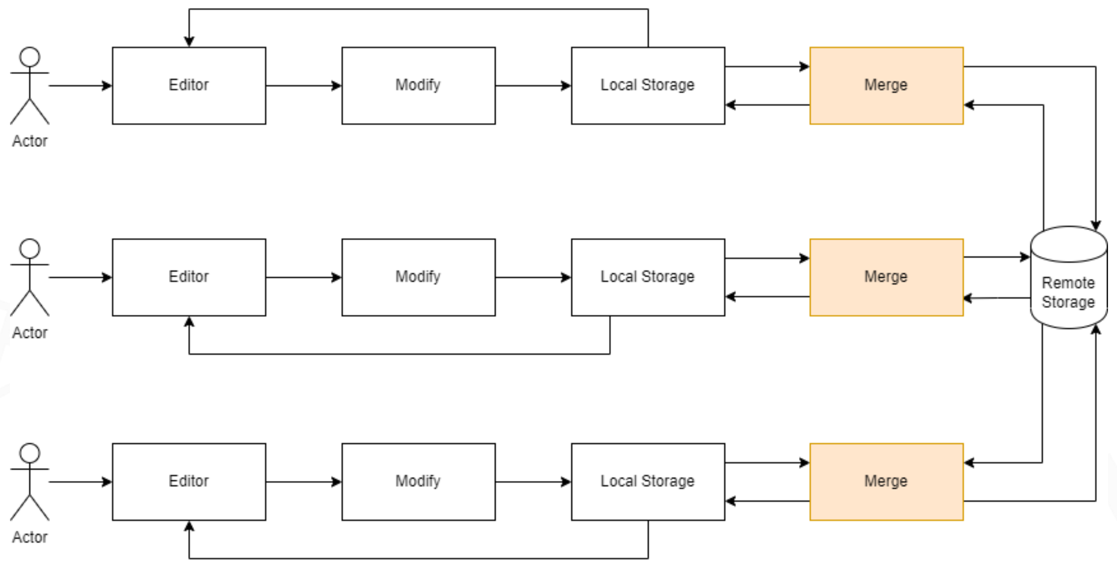
How to Solve These Problems Thoroughly

Operation Transform(OT): Abstract the operation into an operation sequence consisting of an enumerable N atomic operation types

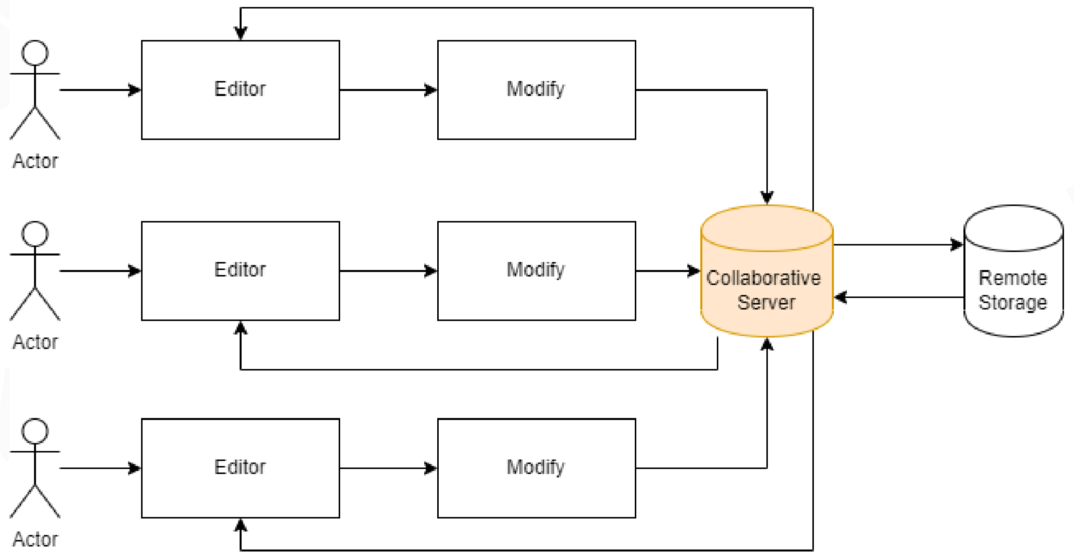
Conflict-free Replicated Data Type(CRDT): A data structure that is replicated across multiple computers in a network, with the following features:

- The application can update any replica independently, concurrently and without coordinating with other replicas
- An algorithm (itself part of the data type) automatically resolves any inconsistencies that might occur
- Although replicas may have different state at any particular point in time, they are guaranteed to eventually converge

Traditional Workflow vs. Collaborative Editing Workflow



Traditional Workflow



Collaborative Editing Workflow

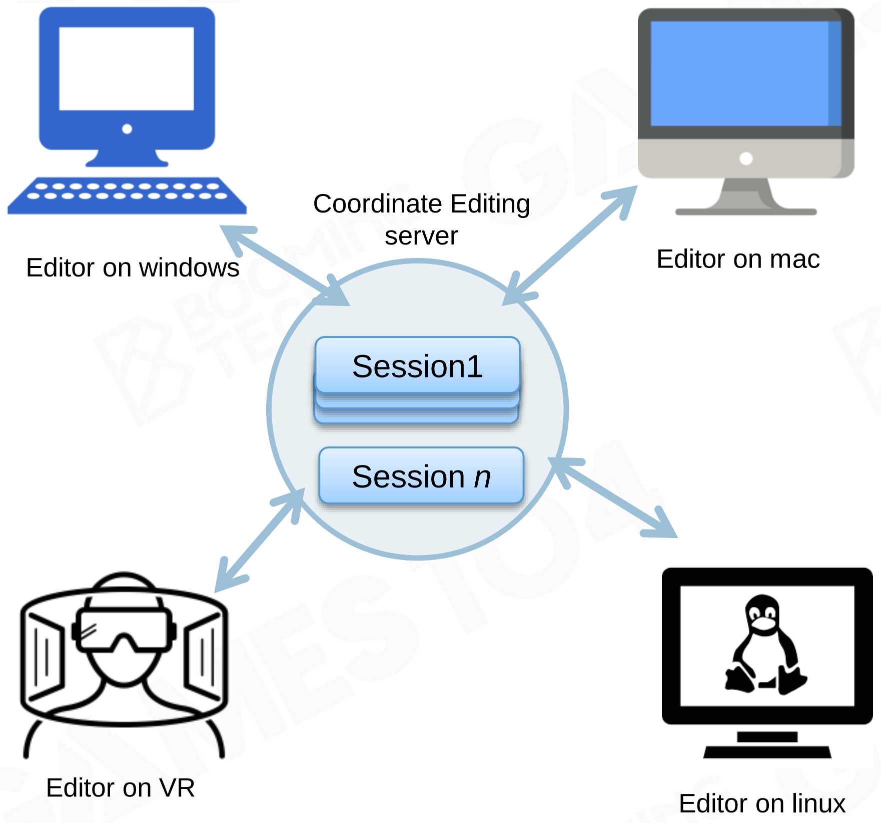
Server is the Most Important Role

Client

- Crash
- Maloperation

Server

- Crash



- The server retains each session until the user who created the session expressly deletes it, or until the server itself is shut down.
- Save session records to disk.

Homework 4(available since 28 June)

- You are supposed to…
  - Comprehend the use of reflection tags
  - Add attributes to the existing system and make them reflect correctly
  - Display new attributes in the detail panel
  - Make them really work in the system(advanced)
  - Write a report document that contains screenshots of your results

- Download at
  - Course-site:

    https://games104.boomingtech.com/sc/course-list/
  - Github:

    https://github.com/BoomingTech/Piccolo/tree/games104/homework04-tool-chains

References

- C++ Reflection, Austin Brunkhorst, 2016:https://austinbrunkhorst.com/cpp-reflection-part-1/
- Clang:https://en.wikipedia.org/wiki/Clang
- Reflective programming:https://en.wikipedia.org/wiki/Reflective_programming
- Unreal Blueprints Visual Scripting:https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/
- Unreal Engine UProperties:https://docs.unrealengine.com/5.0/en-US/unreal-engine-uproperties/
- GPU-Based Run-Time Procedural Placement in 'Horizon: Zero Dawn', Jaap van Muijden, GDC 2017:https://www.gdcvault.com/play/1024700/GPU-Based-Run-Time-Procedural
- Ray casting:https://en.wikipedia.org/wiki/Ray_casting
- Level (video games):https://en.wikipedia.org/wiki/Level_(video_games)

- Unreal One File Per Actor:https://docs.unrealengine.com/5.0/en-US/one-file-per-actor-in-unreal-engine/
- Uses of layers in Unity:https://docs.unity3d.com/Manual/use-layers.html
- Unreal World Partition:https://docs.unrealengine.com/5.0/en-US/world-partition-in-unreal-engine/
- How Figma’s multiplayer technology works, Evan Wallace, 2019:https://www.figma.com/blog/how-figmas-multiplayer-technology-works/
- Unreal Multi-User Editing:https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/MultiUserEditing/
- Conflict-free Replicated Data Types, Marc Shapiro,Nuno Preguiça, Carlos Baquero, Marek Zawirski, 2011:https://pages.lip6.fr/Marc.Shapiro/papers/RR-7687.pdf

- Operational Transformation Frequently Asked Questions and Answers:https://www3.ntu.edu.sg/scse/staff/czsun/projects/otfaq/
- Unreal Sequencer Overview:https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/Sequencer/Overview/
- Unity Timeline:https://docs.unity3d.com/Packages/com.unity.timeline@1.7/manual/index.html
- Plug-in (computing):https://en.wikipedia.org/wiki/Plug-in_(computing)

Lecture 14 Contributor

- Wood	- 爵爷	- Arthas	- 33
- 霓虹甜心	- 令狐冲	- 喵小君	- 小明
- 新之助	- 大喷	- 小鲤鱼	- 乐酱
- BOOK	- Qiuu	- 布鲁布鲁	- 晨晨
- 阿甘	- Adam	- kaiwei	- 怡宝

Q&A

Enjoy ;) Coding

Course Wechat

Follow us for further information