VRUE Tutorial 1

05.10.17

Khrystyna Vasylevska

INTERACTIVE
MEDIA SYSTEMS
Vienna University of Technology

Topics

- Organizational info
- Source Control
- Unity3D and monoDevelop/(Visual Studio)
- Unreal Engine 4 / Blueprints
- Basic Physics
- Leap Motion
- Task 1

Organization

- TISS registration
  - Unregistration deadline 10. October!
- TUWEL course – check your access!
- Equipment hand-out after the Tutorial till 5PM
  - Only for full registered groups!
  - Read and fill-in the contracts!
  - Bring you Student ID + 1 copy of it
- Please keep your repositories **private for all assignments at all times**!

Organization (2)

- Check your PC+Vive **ASAP**!
  - Make sure that base-stations are fixed rigidly!
  - Moving base-stations when powered might damage them!
- Please, handle the packaging with care!
  - Some of it is falling apart -> can damage the equipment
- If something is not working and tech. support doesn't help – let us know asap!

Organization (3)

- TUWEL course
  - Tutorial slides
  - Group registrations
    - Those without a group – will be assigned!
  - Engine choice
  - Assignment descriptions
  - Assignment submissions (up to 256MB)
  - Forum (tutors are there for you!)
  - FAQ

- Contact us: vrue@lists.tuwien.ac.at

Source-Control Bitbucket

- You can have unlimited **private** repositories under certain conditions.
  - It is free as long as you have fewer than 5 collaborators registered.
  - You can always manage and remove collaborators temporarily and add them later on.

Source-Control GitHub

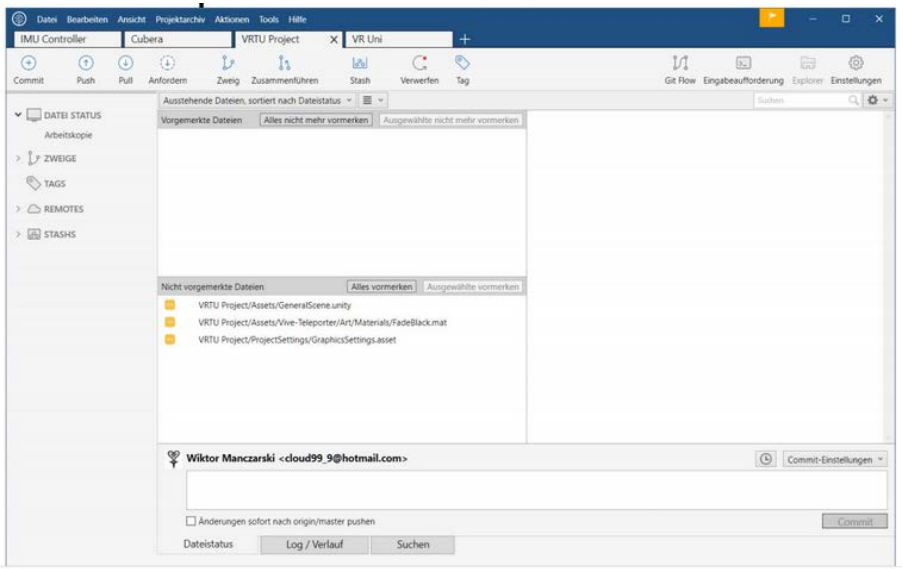
- If you are using GitHub, we kindly ask you to register as a student. At GitHub it is free!
- You'll need a TU email address + valid student ID
- Then you can hide your code from the public
  - Please, do that with all the assignments.

Source-Control .gitignore

- It is recommended to use a .gitignore file for Unity and Unreal.
  - It ignores Source-Control for certain files
  - Unity and Unreal have many auto-generated files that don't need to be in the repository.
  - Can make the repository huge (Unreal generates >2GB of files even though only around 300MB need to be shared with your partner)
- Check gitignore.io for templates
  - https://www.gitignore.io/

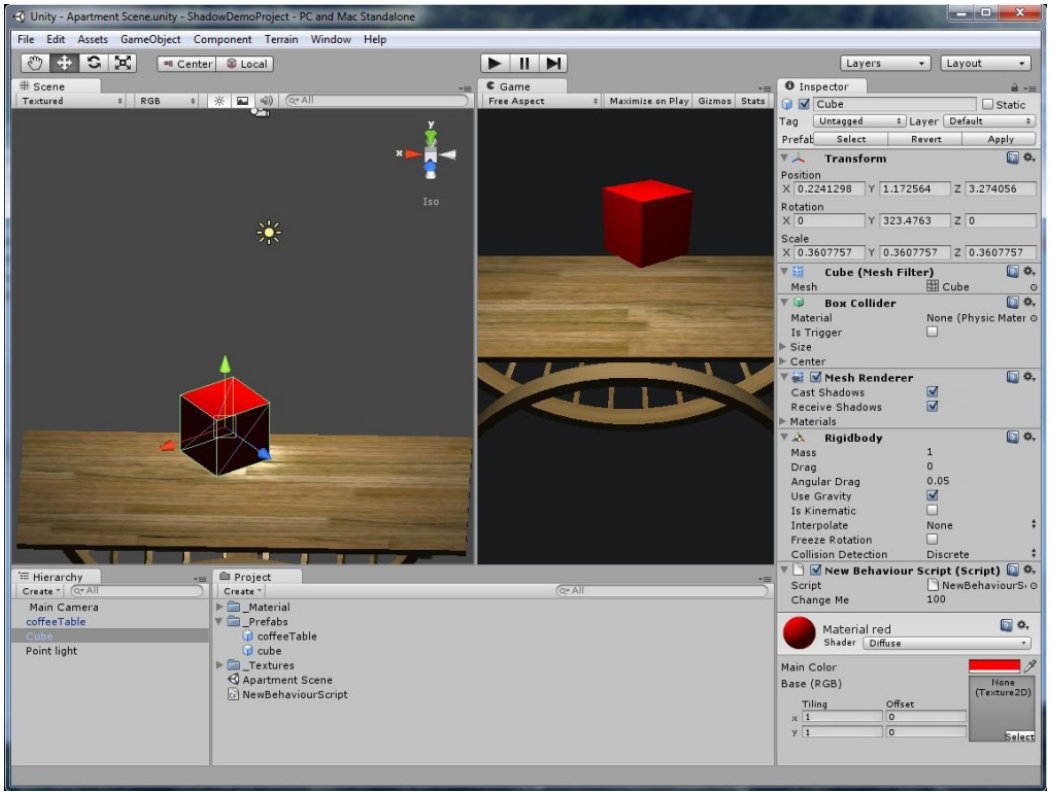
Source-Control git Client

- You can either use the git Console or as an alternative a Client like SourceTree
  - Very easy to use
  - Graphical user Interface



Unity3D
+
MonoDevelop/Visual Studio

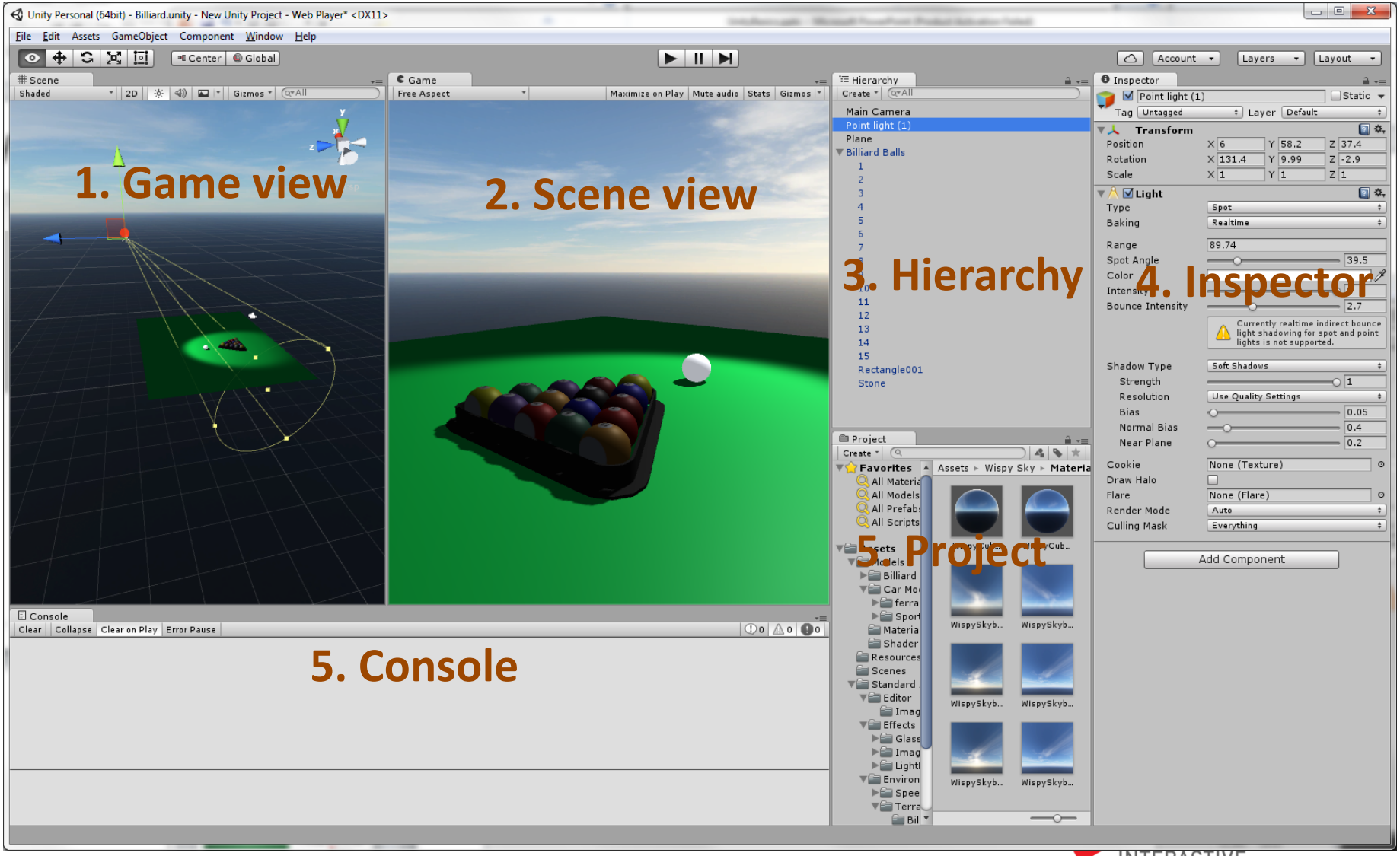
Unity 3D



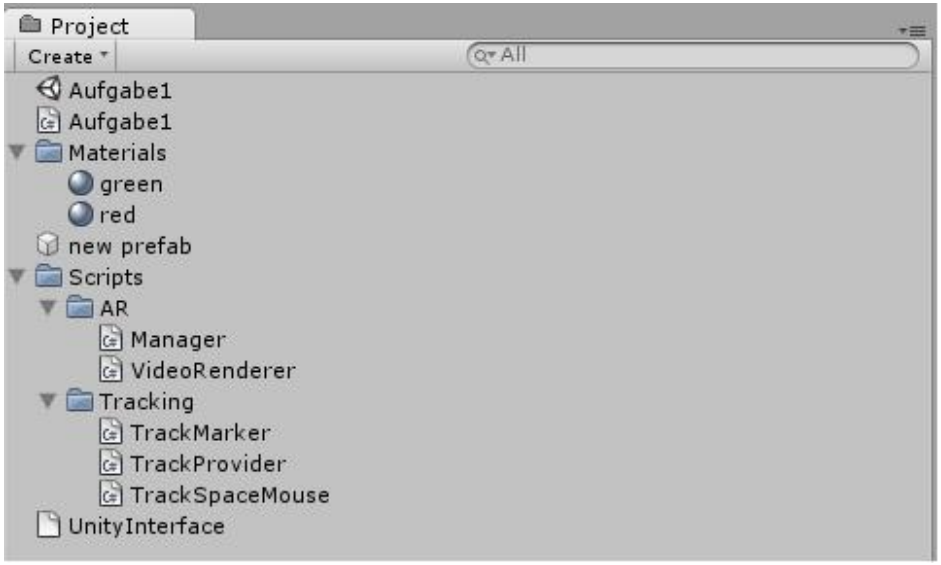
- Game Engine
  - Adapted for AR&VR
- Development environment
  - Editor
- Runtime actor - Player
- Supports many media formats
  - 3D models
  - Sounds
  - Animations
- Programming
  - C# (Mono) – relevant for us
  - JavaScript
  - Etc.

https://docs.unity3d.com/Manual/UnityOverview.html

Interface


1. Game view
2. Scene view
3. Hierarchy
4. Inspector
5. Project
5. Console

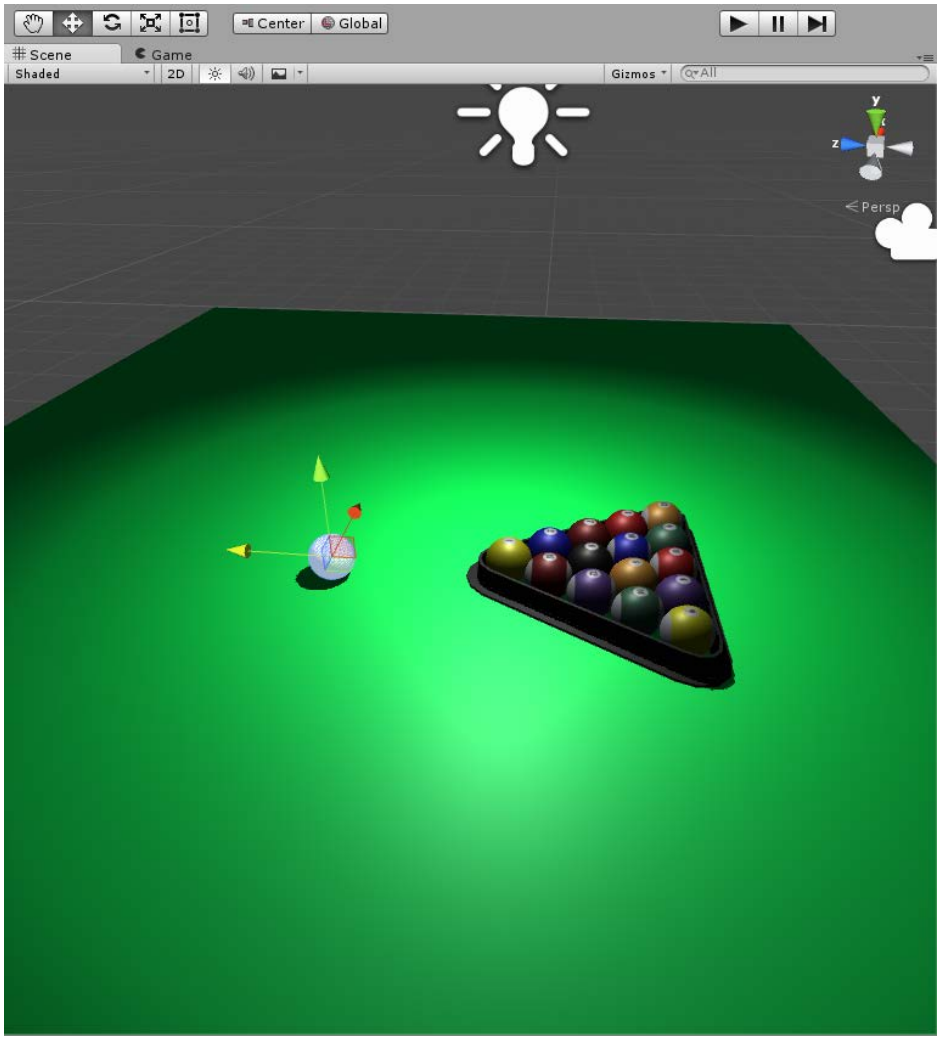
Unity – Project tab



- Assets
  - Unity Ressources
    - Meshes
    - Scripts
    - Textures
    - Scenes
    - ...
  - Direct connection to Filesystem
  - Organization depends on developer

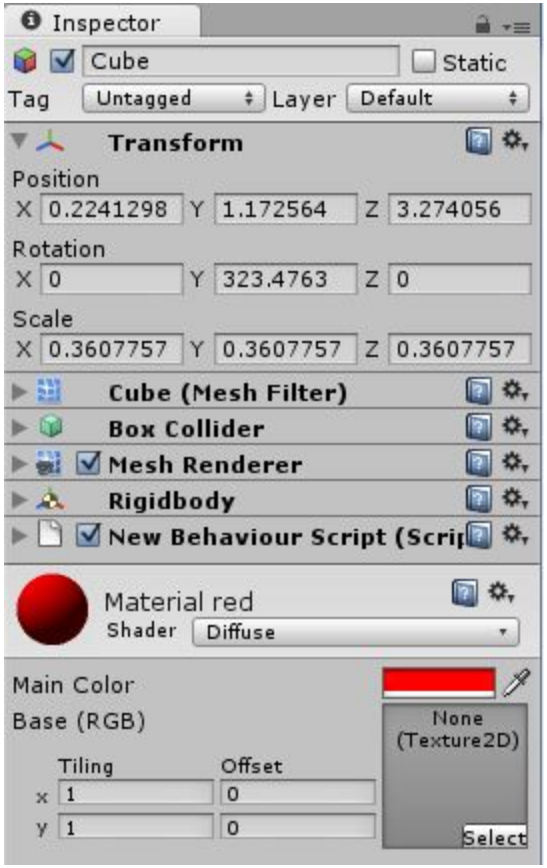
Unity – Scene & Game tabs

QWERT



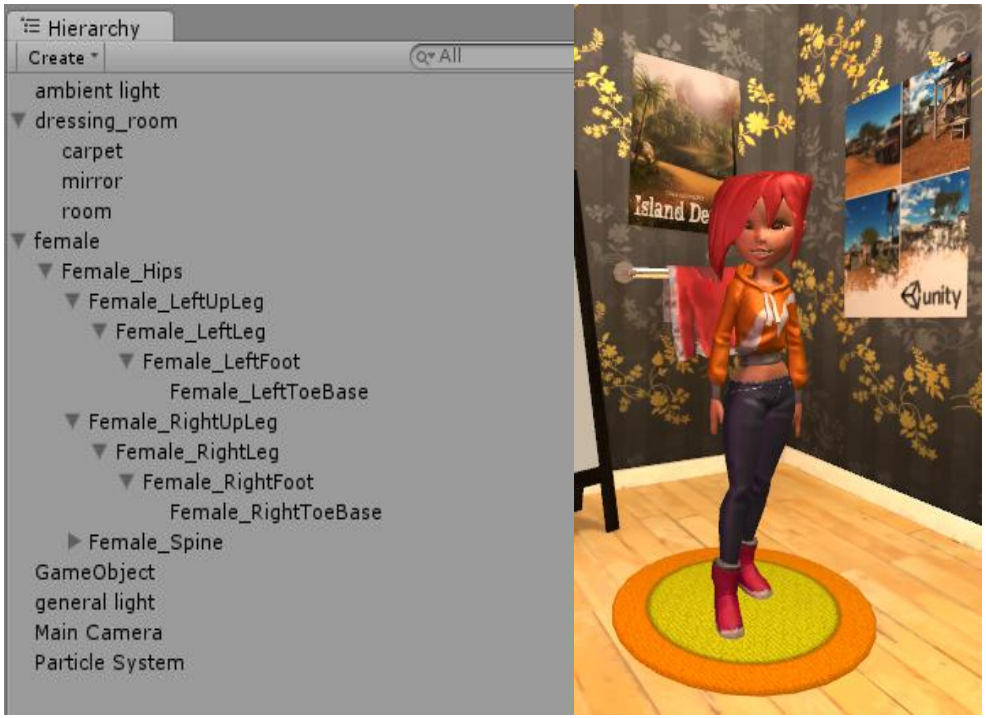
- Scene - Visual construction area
  - Allows visual manipulation
    - Position
    - Rotation
    - Scale
  - Detailed view
    - Wire frames
    - Alpha chanel...
  - Partial previews
    - Sound (depending on vie point)
    - Some animations (particle systems)
- Game - Real time player view
  - Multiple cameras view is possible
  - Sounds
  - Player triggered events
  - All changes done are NOT saved!

Unity - Inspector



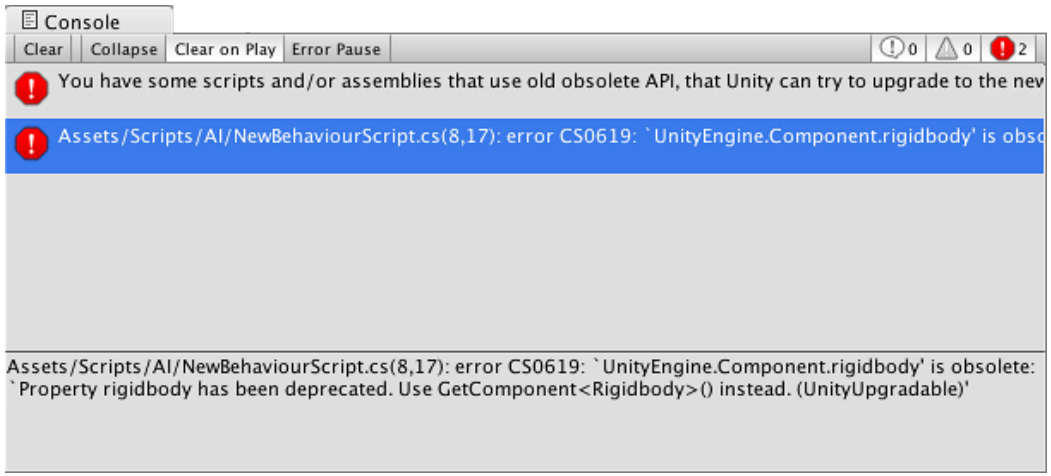
- GameObjects
  - All Objects in the Scene
  - Container for Components
  - Can be deactivated, taged, assigned to a layer
  - Every Object has a Transform Component
- Components
  - Define the functionality of GameObjects
  - Differ in Types
  - Are attached to GameObjects
  - Can be added
    - in Editor via Menu or Inspector button
    - attached by a Script
    - Using Drag and Drop from Project
  - Script is also a Component

Unity – Hierarchy tab



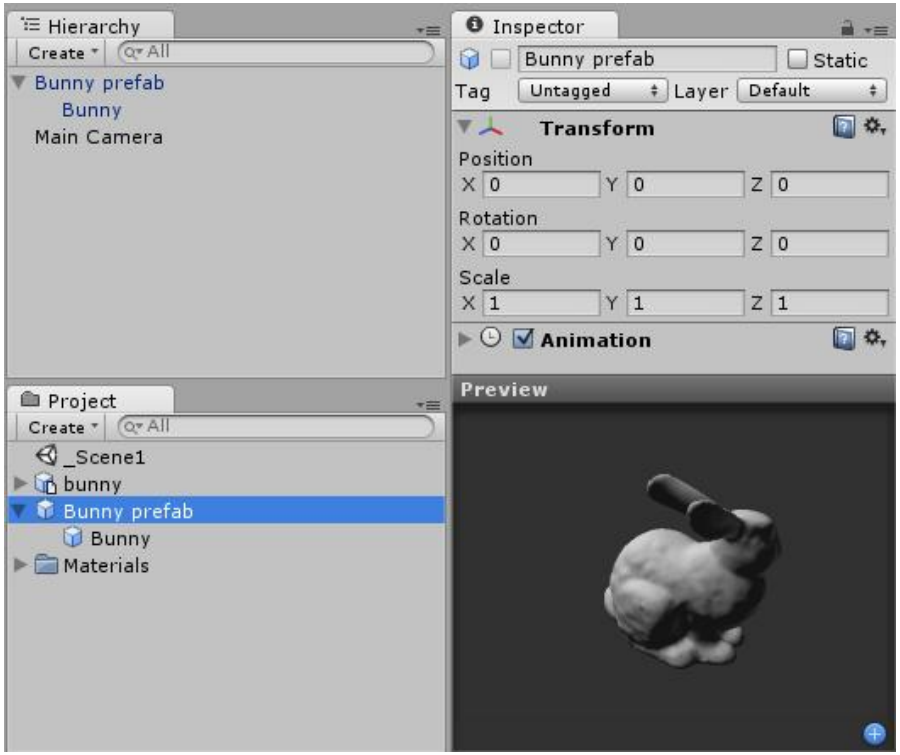
- Defines the scene organization
- Showes all objects in the current scene
- Defines Parent-Child relations
  - Defines grouping of GameObjects
  - Builds up on Transform Component
- Parent
  - is a local coordinate system for Children
  - Influences Children‘s Properties

Unity - Console



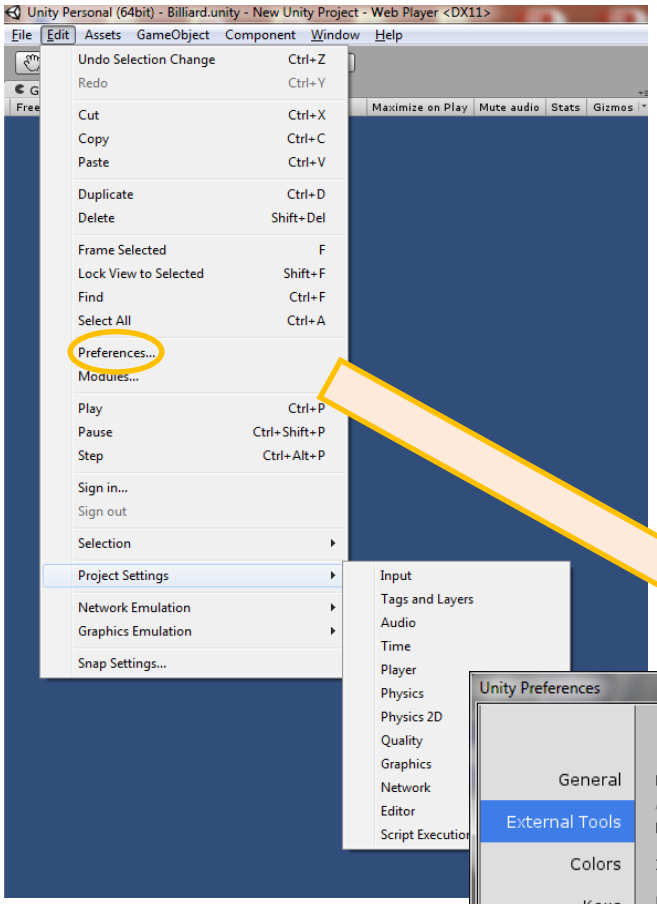
- Errors/warnings
  - Obsolete elements warnings
  - Unused variables warnings
- Your script messages using
  - Debug.Log
  - Debug.LogWarning
  - Debug.LogError
- Features:
  - Message separation
  - Collapsing messages from the same line
  - Clear on Play

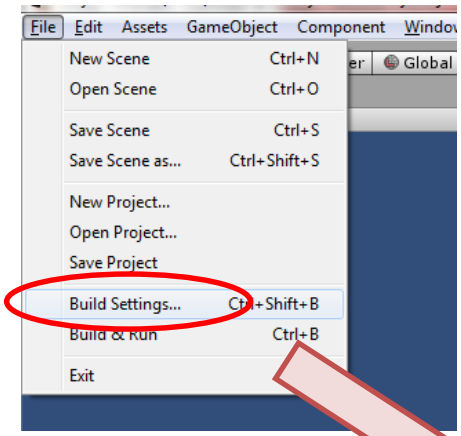
Prefabs

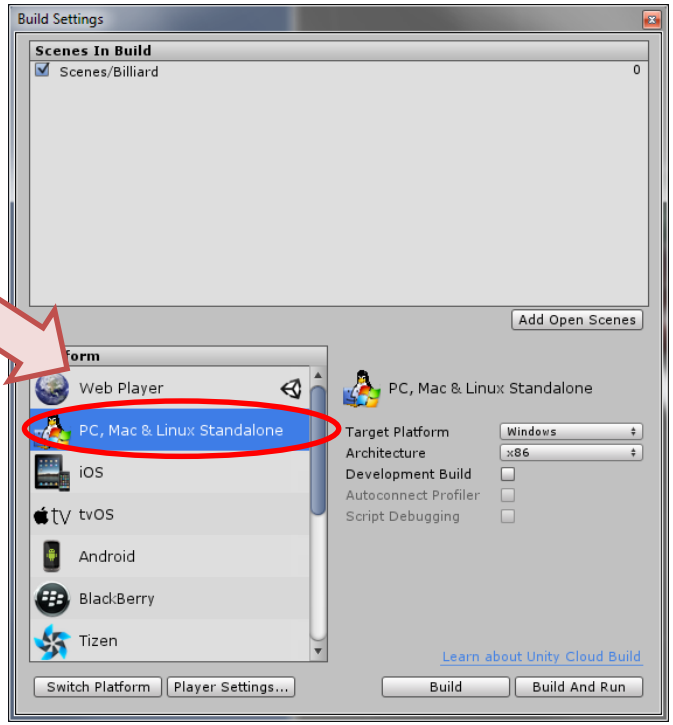


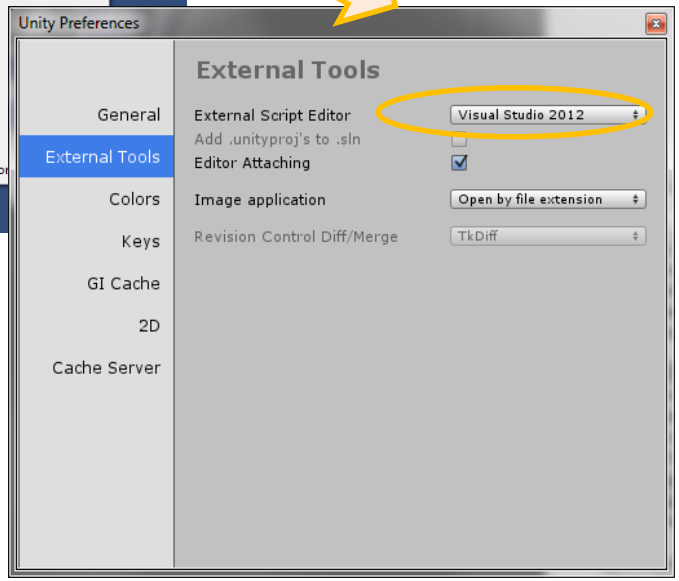
- Preconfigured GameObjects
  - Can be instantiated or cloned
    - Runtime
    - Reused in different Scenes
  - Already contain chosen
    - 3D model
    - All Components
    - Pre-defined settings of Components
    - Pre-defined values of public variables in Scripts
  - You can edit
    - Instance of a Prefab
    - Apply Instance Settings to the Prefab (save them)
    - Prefab itself
  - You can create Prefabs during runtime too

Settings and Preferences

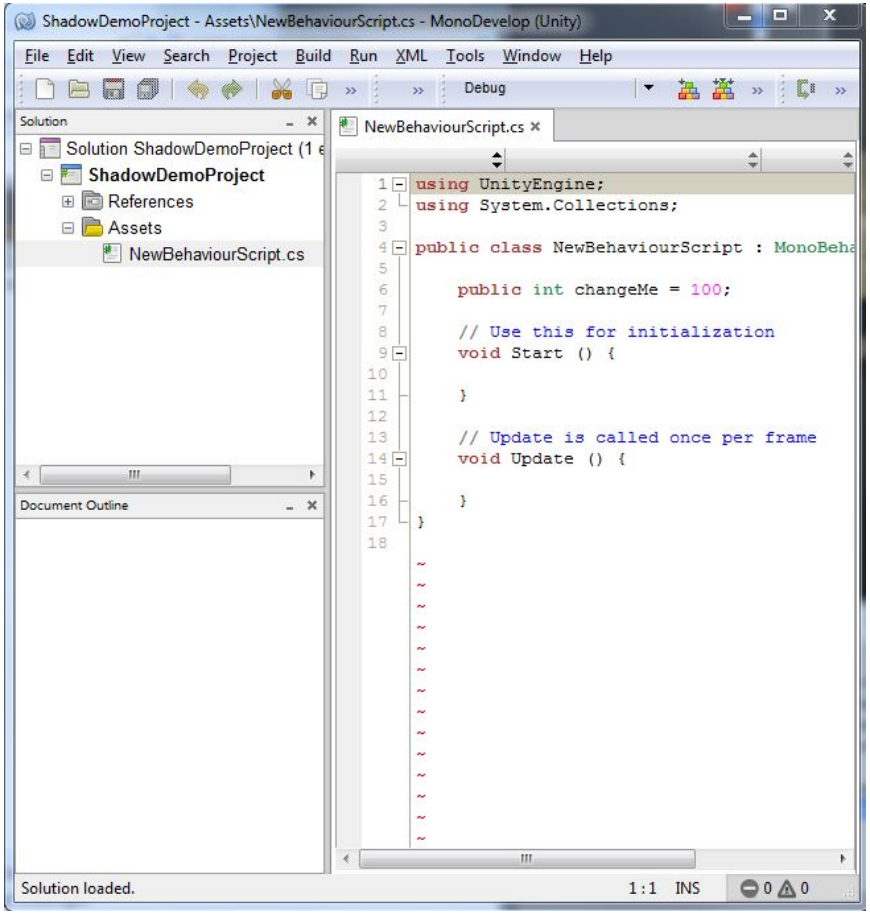








MonoDevelop



- Integrated Development Environment
  - Syntax Highlighting
  - Debugging
  - Good Unity-Integration
  - Project management support
  - …
- Feel free to use Visual Studio
- Scripts
  - Assets
  - automatischer Build-Prozess im Editor

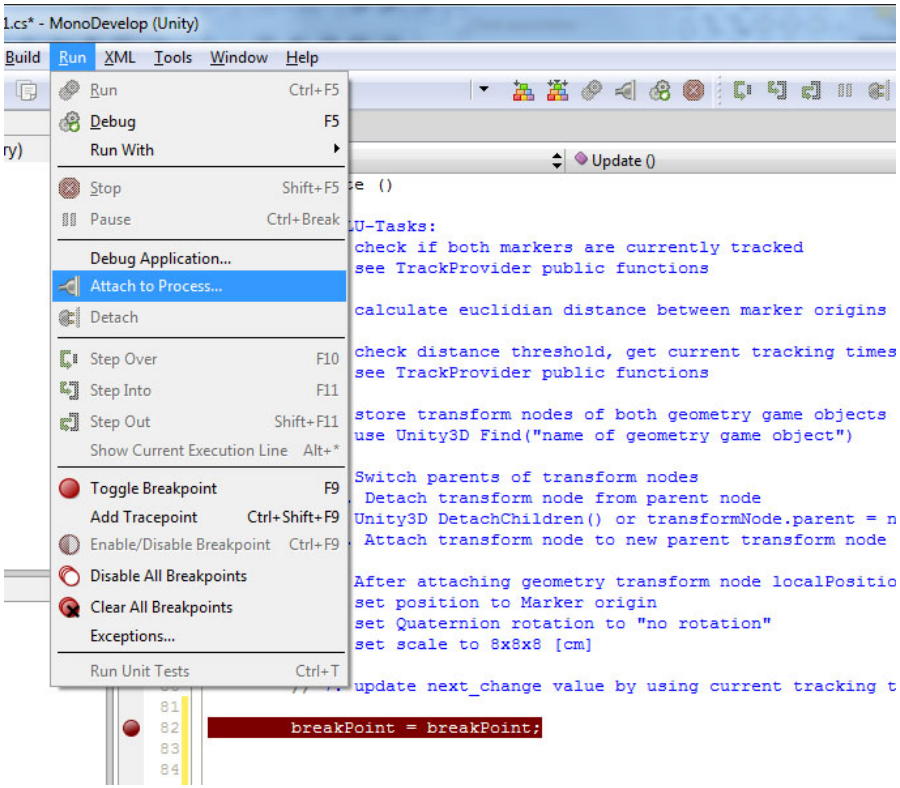
Basics

- `MonoBehavior`
  - Is attached as Component to a GameObject
  - main Methods in Script
    - `Start()` - Initialisation
    - `Update()` – called on every new frame
  - Access to GameObject & Components
    - `gameObject`
    - `transform`
    - `GetComponent<T>()`

To Keep in Mind

- `new GameObject()`
  - Creates new GameObjects in the root of Hierarchy (parentless)
- `transform.parent`
  - Setting the parent maintains the current Object's **global world values**
    - `transform.position`,
    - `transform.rotation`,
    - `transform.scale`
  - But **local transform values** will be corrected in regard to the new local coordinate system (the one defined by the parent)!
    - `transform.localPosition`,
    - `transform.localRotation`,
    - `transform.localScale`

MonoDevelop-Debugging



- before Starting the Scene attach MonoDevelop to the Unity-Instance
- Set Breakpoints
- Start the scene in Unity

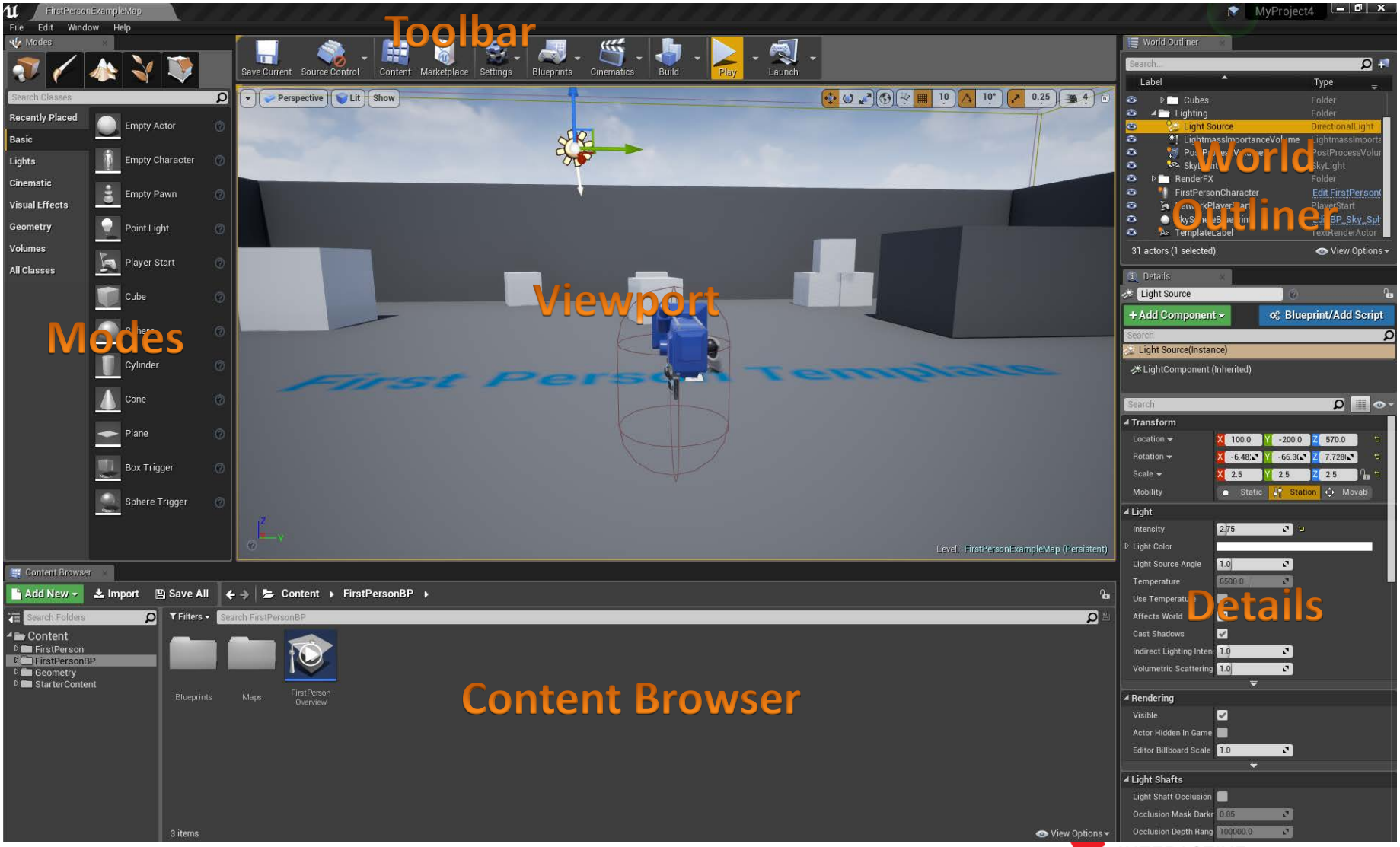
Unreal Engine 4

Unreal Engine 4

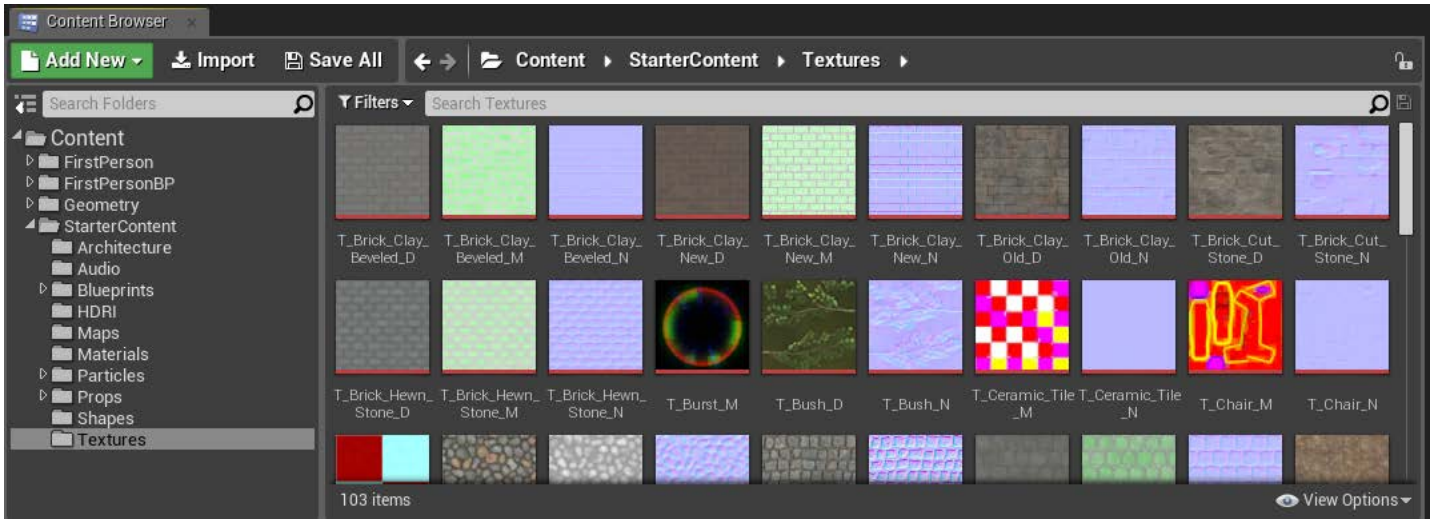
- Game Engine
- Development environment
  - Unreal Editor
- Runtime actor – Player
- Supports many media formats
  - 3D models
  - Sounds
  - Animations
- Programming
  - C++ (Visual Studio)

https://docs.unrealengin.com/latest/INT/

Unreal Editor Interface

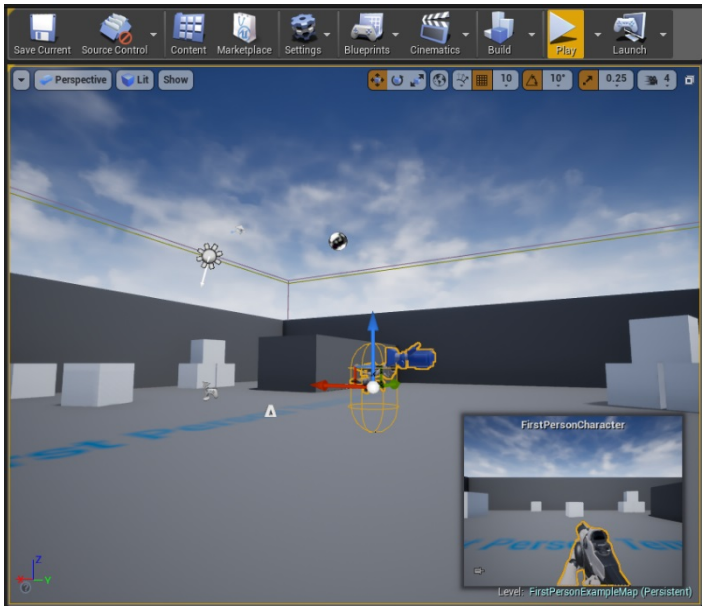


Unreal – Content Browser

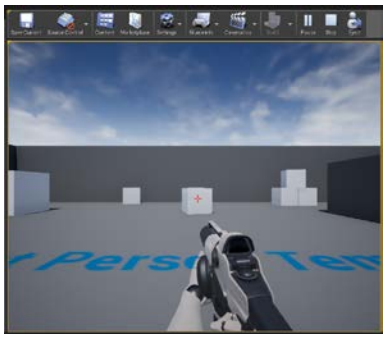


- Assets
  - Create, import, organize and modify content
    - Meshes, Textures, Blueprints,…
  - Whether to show engine and plugin content can be toggled via the view options

Unreal – Viewport



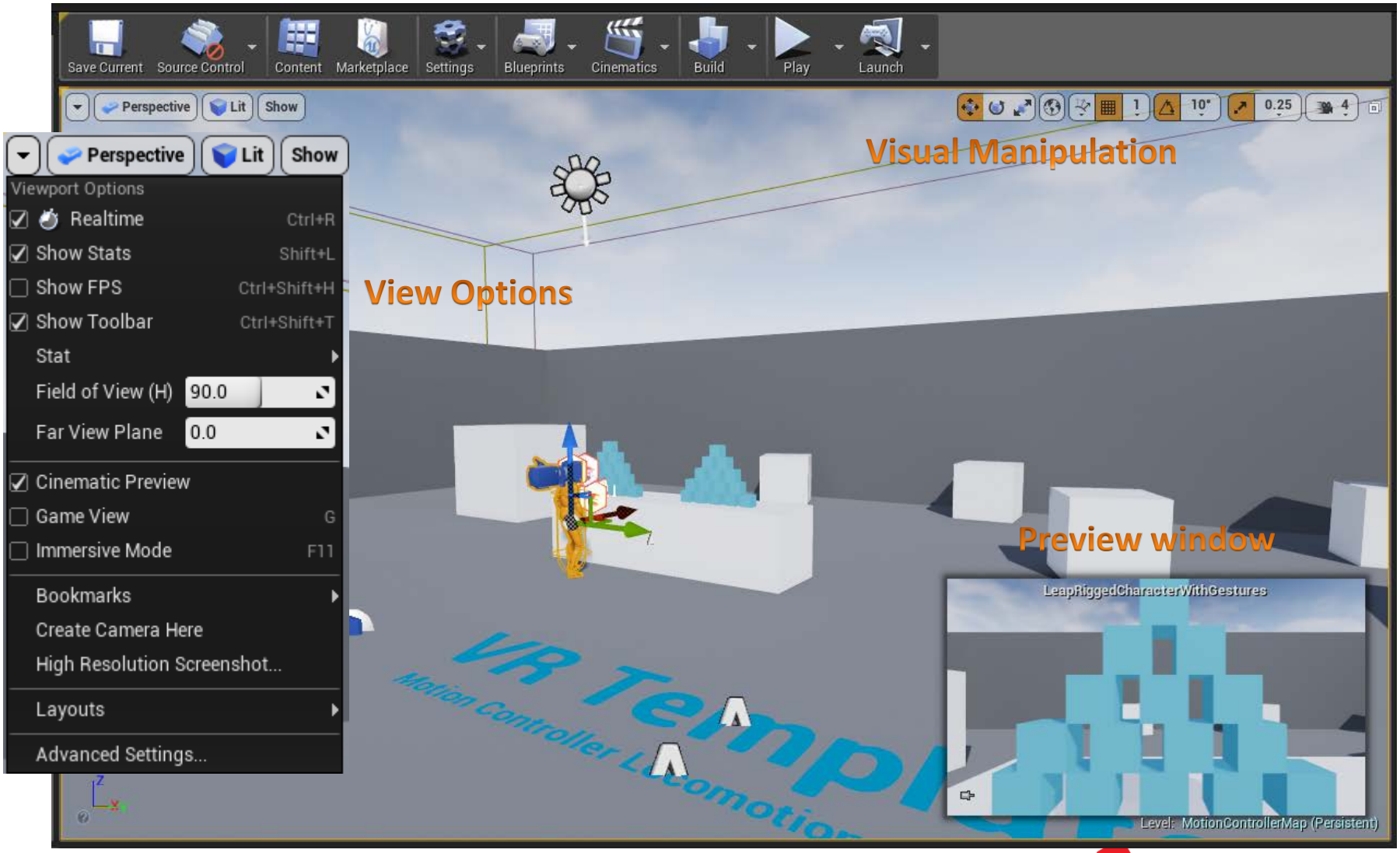
- Visual construction area
  - Allows visual manipulation
    - Position, Rotation, Scale
  - View Options
    - Perspective
    - Lit, unlit, wireframe, …
  - Preview window
    - When player character selected



- Preview: Real time player view
  - Press play to start preview in viewport
  - Option for VR Preview

For more information on viewports look at:
https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/Viewports/index.html

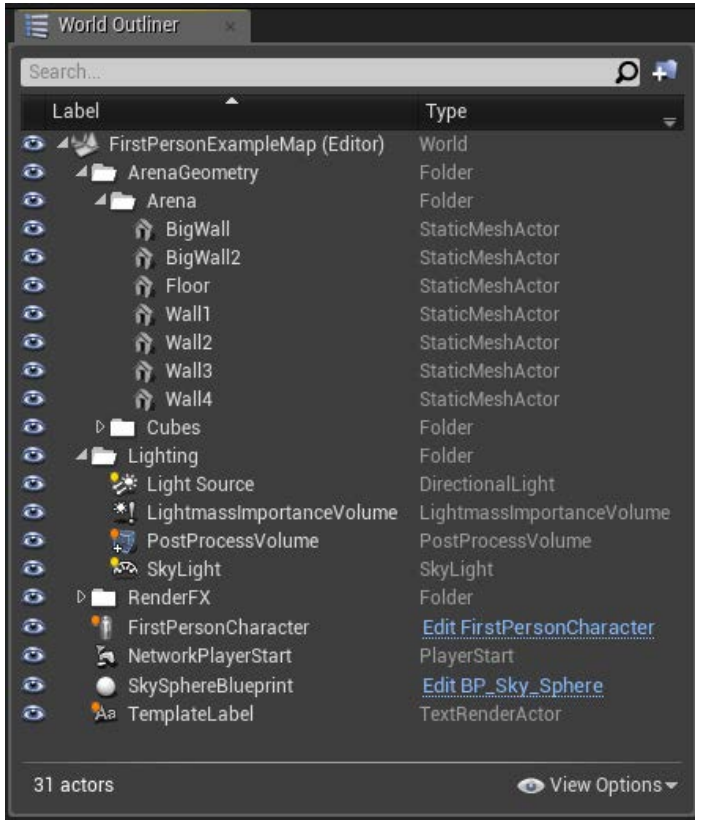
Unreal – Viewport



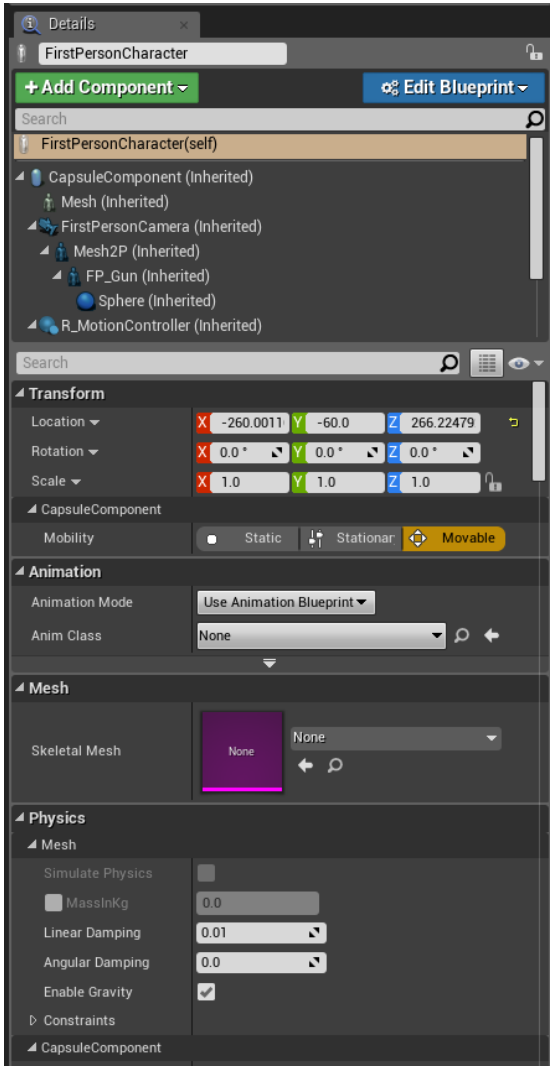
For more information on viewports look at:
https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/Viewports/index.html

Unreal – World Outliner

- Shows all Actors in the scene and their types
- Hierarchical tree view
- What you can do here
  - Search for specific actors
  - Toggle visibility
  - Select actors to modify
  - Group actors
  - Attach an actor to another actor
  - Focus on an actor by pressing **F**
  - Find asset in content browser
  - Organize actors in folders
  - Access blueprint editor for blueprint types
  - …



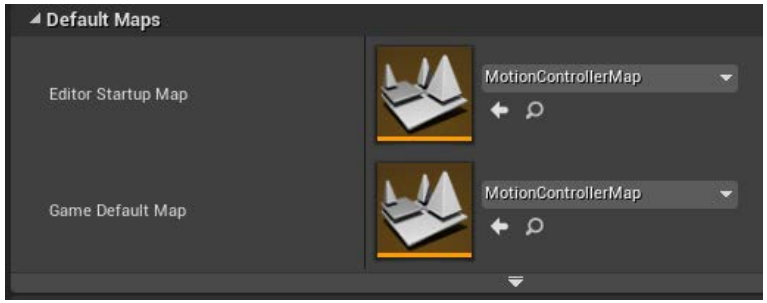
Unreal – Details Panel



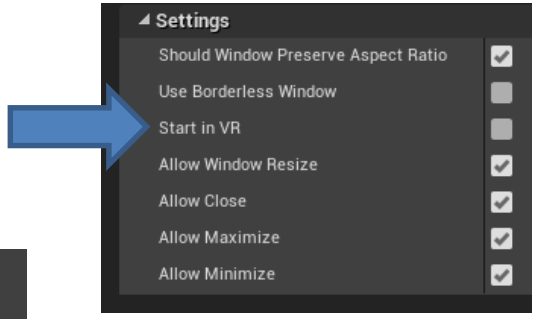
- Details for the actor selected in the Viewport

Settings and Preferences

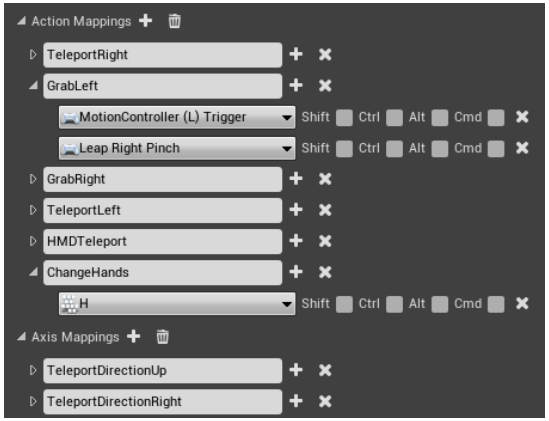
- Settings->Project Settings
  - Maps & Modes
    - Editor Startup Map
    - Game Default Map (always make sure this is set to the map you want before packaging a project)



  - Description
    - Start in VR (if your program uses VR, don‘t forget to check this)



  - Input
    - Add ActionMappings and AxisMappings



More information on input and axis mappings:
https://www.unrealengine.com/en-US/blog/input-action-and-axis-mappings-in-ue4

UE4: Actor

- An **Actor** is any object that can be placed into a level
  - Called GameObjects in Unity
  - There are several different types of Actors, some examples include: StaticMeshActor, CameraActor, and PlayerStartActor

- Creating a new instance of an actor is called **spawning**
  - Function in blueprints `Spawn Actor`

- Contains the Root Component
  - stores main transform (location, rotation, and scale)
  - applies its properties to children

For more information on Actors and Components:
https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Actors

Actor Classes

- An **Actor** is an object that can be placed or spawned in the world.
- A **Pawn** is an Actor that can be "possessed" and receive input from a Controller.
- A **Character** is a Pawn that includes the ability to walk, run, jump, and more.
- A **Player Controller** is an Actor responsible for controlling a Pawn used by the player.

UE4: Actors & Components

- **Actors** can be thought of, in one sense, as containers that hold special types of **Objects** called Components.
  - **ActorComponent** is the base class for components that define reusable behavior that can be added to different types of **Actor**s.
    - are associated with a specific Actor, but do not exist at any specific place in the world
    - E.g. AI, deal with player input
  - **Scene Components**
    - Actor Components with transforms
    - can be attached to each other in a hierarchical fashion
  - **Primitive Components**
    - Scene Components with rendered representation
    - E.g. mesh, particle system, physics and collision settings are here

For more information on Actors and Components:
https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Actors

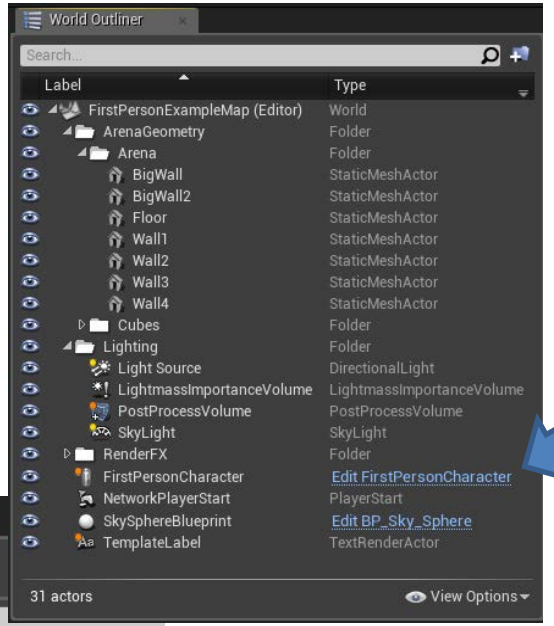
Blueprints Visual Scripting

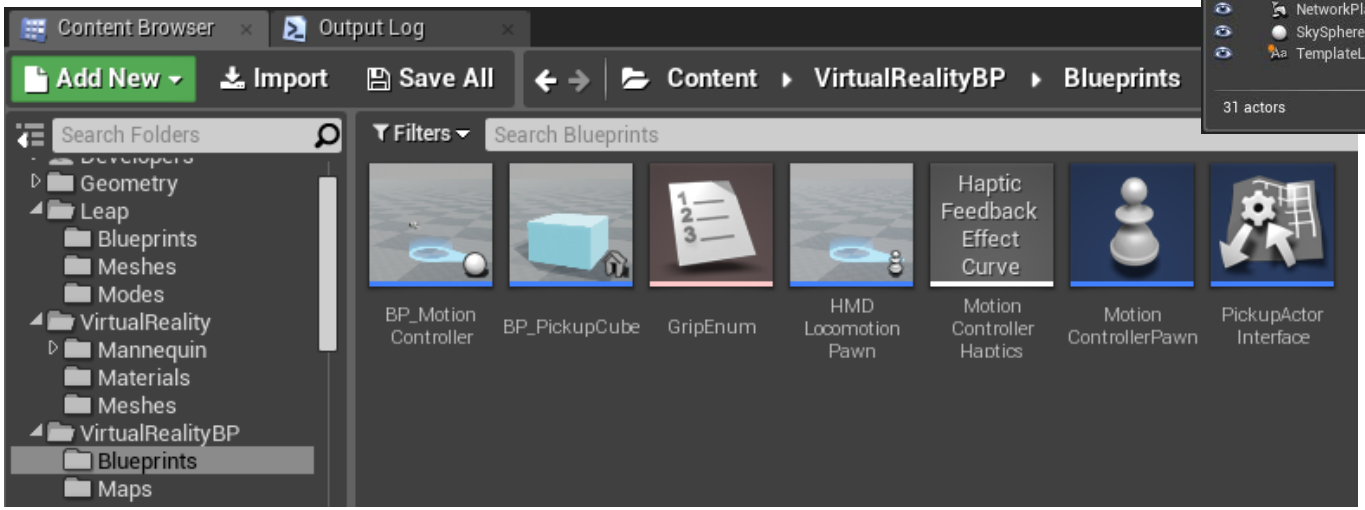
- The **Blueprints Visual Scripting** system in Unreal Engine
  - is a complete gameplay scripting system
  - based on the concept of using a node-based interface
  - is used to create gameplay elements from within Unreal Editor

Getting started with Blueprints:
https://docs.unrealengine.com/latest/INT/Engine/Blueprints/GettingStarted/index.html

Blueprint Editor Access

- In World outliner Type in blue
- In Content browser

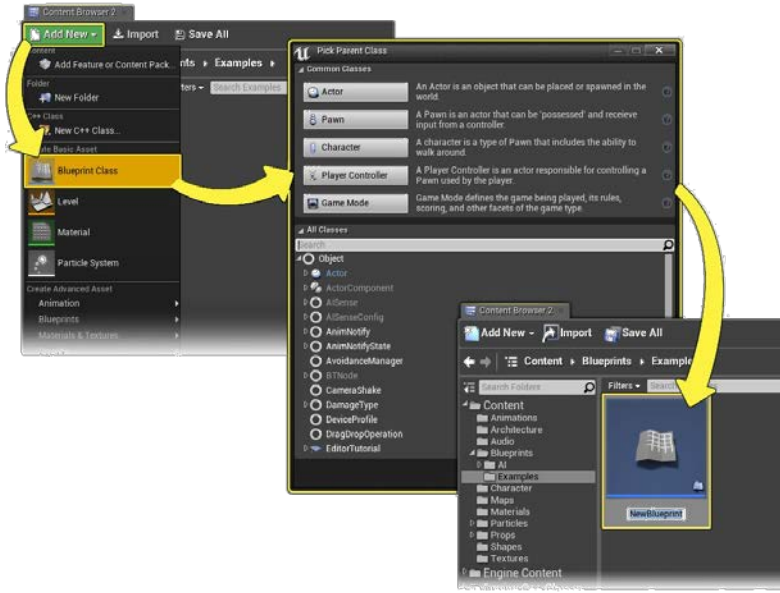




Blueprint Types

- Level Blueprints
  - are specific to the level that they're used in
  - Are used to set up functionality specific to the level, or Actors in it

- Blueprint Classes
  - Define reusable behavior in the project
  - Are defined by parent blueprint class
    - add it to any of your levels
    - also add as many copies as you need



More information about the level blueprint can be found here:
https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/LevelBlueprint

UE4: Blueprint Classes

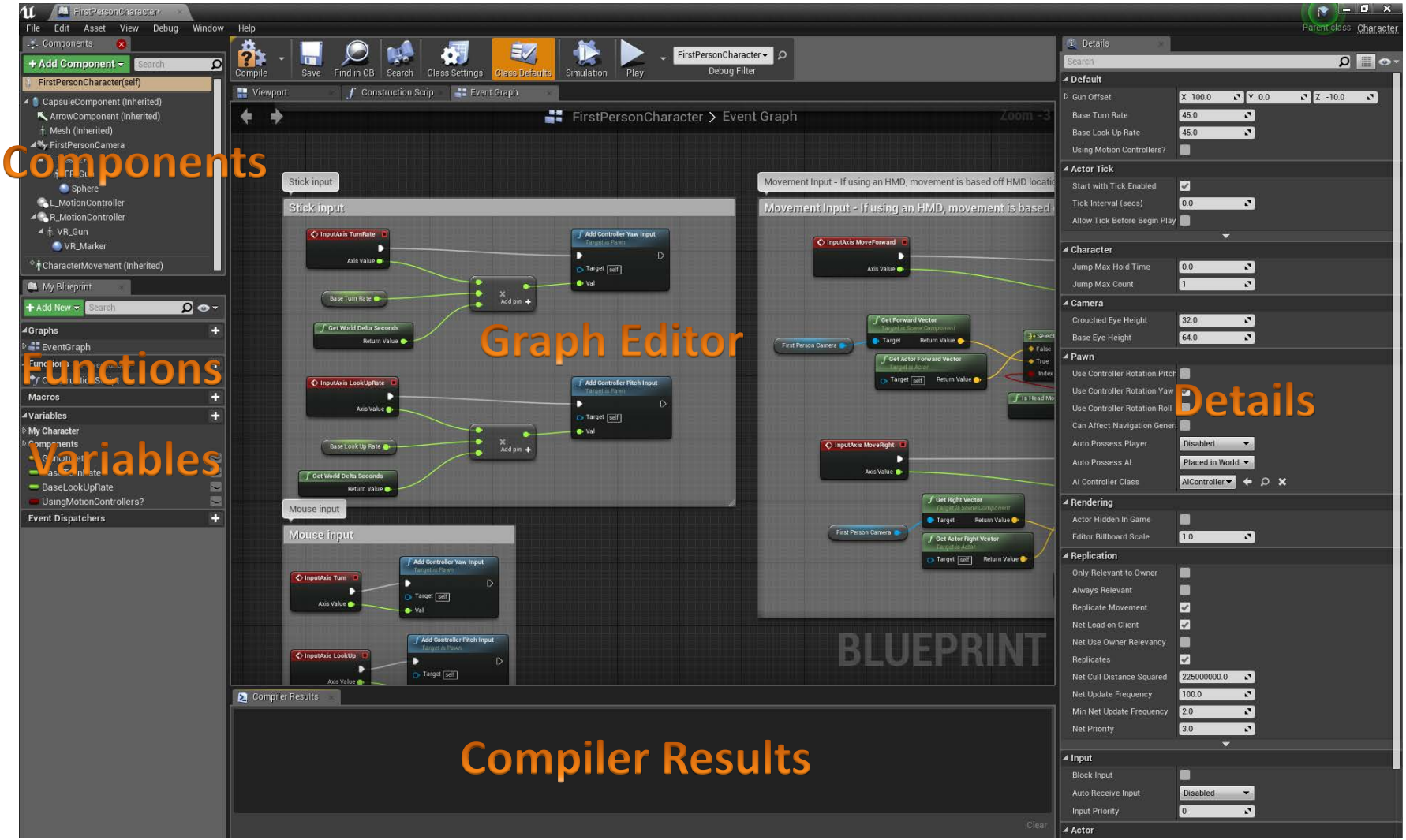
- Similar to classes in Java/C++ (inheritance, interfaces)
- Can be instantiated (*spawned*) at Runtime

- Can contain
  - Variables, Functions
    - Mesh, Material, etc.
    - Components with Pre-defined settings

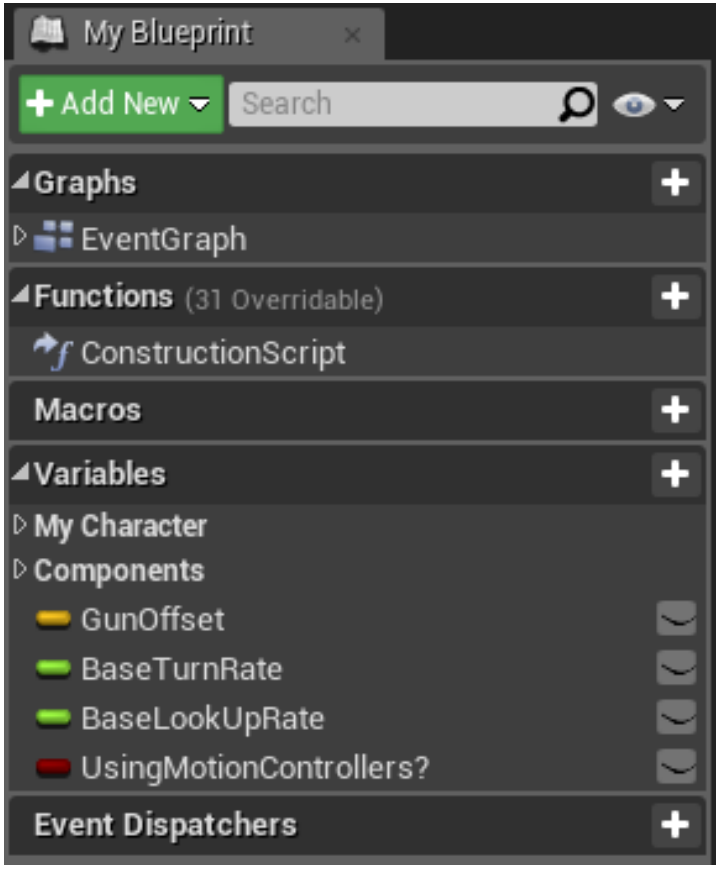
- You can edit
  - Blueprint Class in general (applied to all Actors in project)
  - Instance of a Blueprint Class (applied to a specific Actor)
    - Make sure you are working on the one you want!

Quickstart tutorial on Blueprints:
https://docs.unrealengine.com/latest/INT/Engine/Blueprints/QuickStart/index.html

The Blueprint Editor

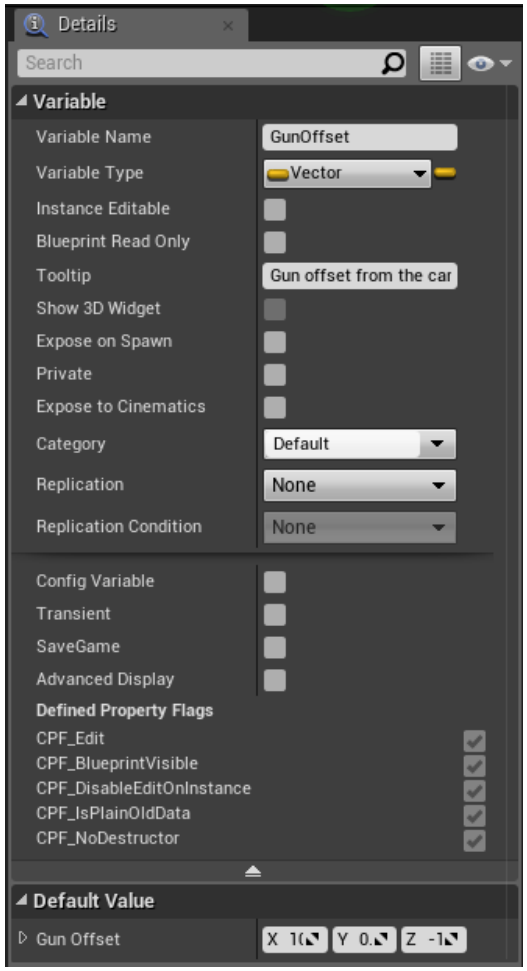


Blueprint Editor: My Blueprint



- The items displayed depend on the type of the Blueprint
  - Graphs
  - Functions
  - Macros (template for nodes)
  - Variables
  - Event Dispatchers
- create new items by clicking on the plus icon
- Variables can be set to private or public (eye icon)

Blueprint Editor: Details



- Edit properties of the blueprint
  - Variables
  - Functions
  - Events
  - Actor components

Blueprints: Accessing Data

- Inside the current Blueprint
  - **Drag** functions/components/variables/… into the Graph Editor to use them
  - Or right click in the Graph Editor and **search** for the functions/component/variable

- Actors from the World Outliner can be accessed in the **Level Blueprint**
  - you can drag an actor from the World Outliner into the Level Blueprint
  - Every level has a Level Blueprint
  - To access the level blueprint select Blueprints > Open Level Blueprint

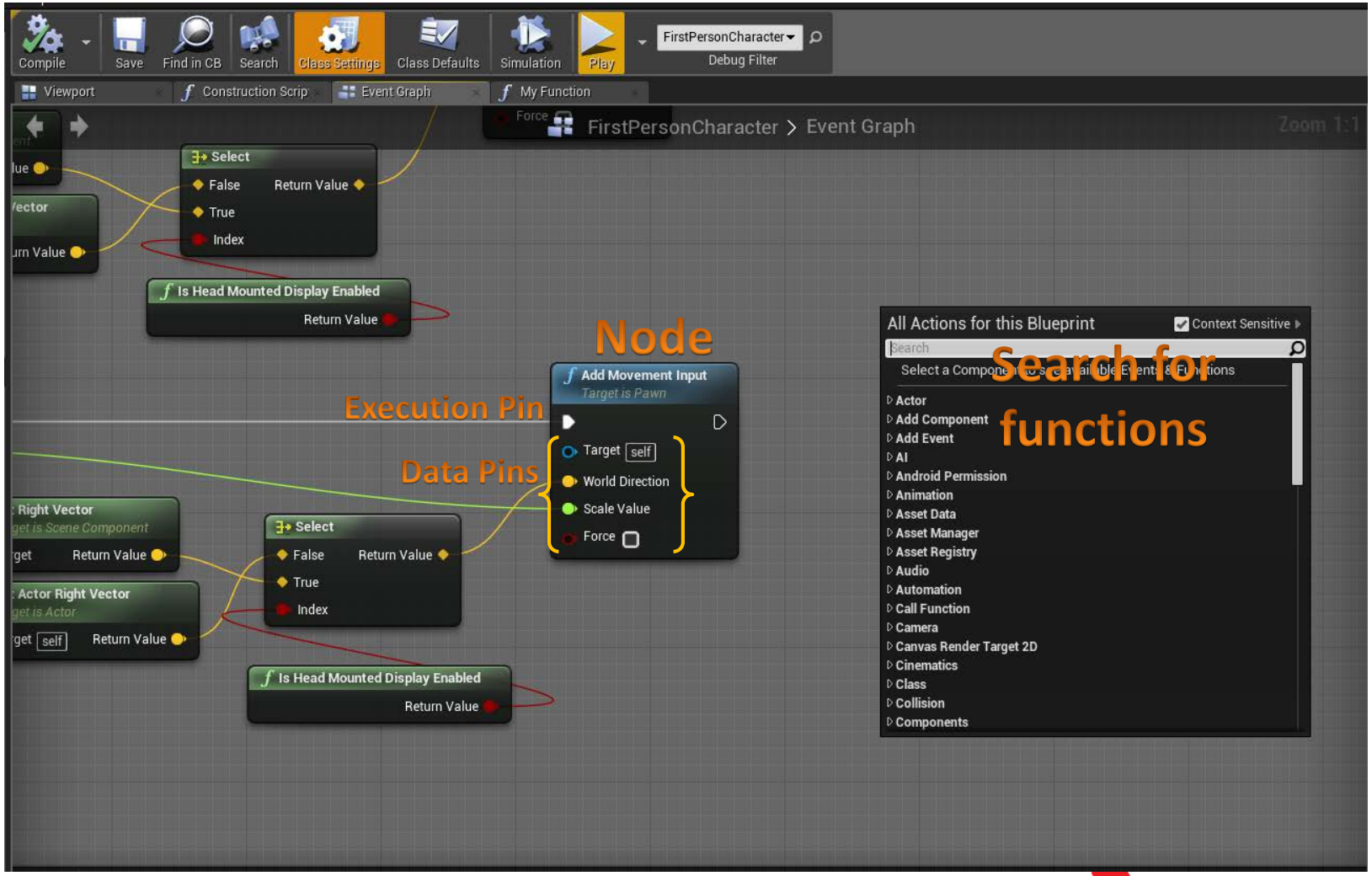
More information about the level blueprint can be found here:
https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/LevelBlueprint

Blueprints: Accessing Actors

- Access Actors not associated with the current blueprint
  - `Get All Actors Of Class`
  - `Get All Actors With Tag`
    - These are slow, don‘t use them every frame!
  - Use various Tags and `Get Tags` for faster search

- Useful information:
  - Referencing Actors in Blueprints
    - https://docs.unrealengine.com/latest/INT/Gameplay/HowTo/ReferenceAssets/Blueprints/
  - Finding Actors in Blueprints
    - https://docs.unrealengine.com/latest/INT/Gameplay/HowTo/FindingActors/Blueprints/index.html

Blueprint Editor: Graph Editor



Blueprint Editor: Graph Editor

- Visual scripting
  - Right click to add a node
    - context sensitivity available
  - Add comments to sections for readability
    - type comment in search function, select *add comment*
    - Or select some nodes, right click on one of the nodes, and select *create comment*
  - Search functionality
    - Use to look up available functions/variables

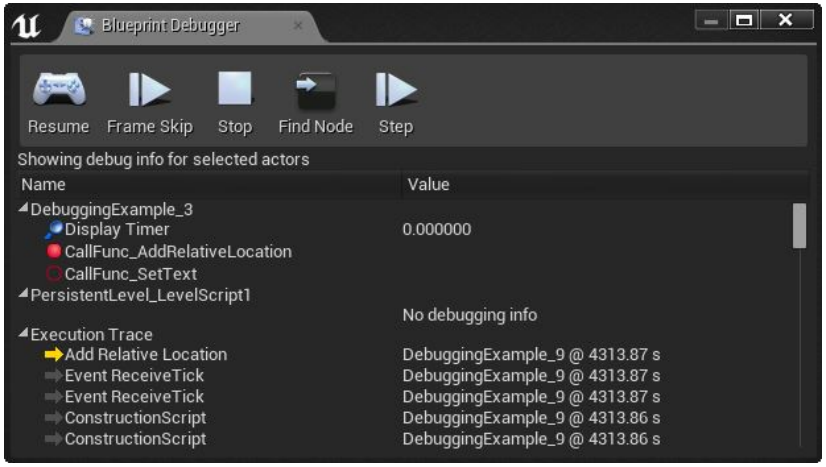
More on Pins:
https://docs.unrealengine.com/latest/INT/Engine/Blueprints/BP_HowTo/ConnectingNodes

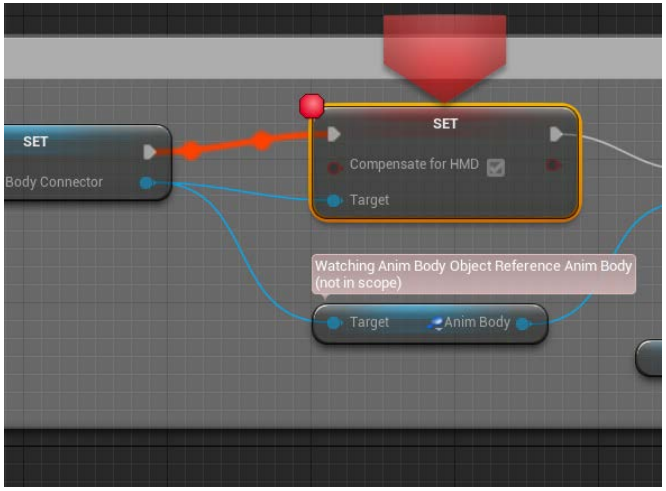
Blueprints: Useful Tips

- `Spawn Actor From Class`
  - Create an Actor at runtime
- `Destroy Actor`
  - Destroy an Actor at runtime
- `Get Actor Transform`
- `Attach To Actor/Component`
- `Detach From Actor/Component`
- Blueprints Quick Start Guide
  - https://docs.unrealengine.com/latest/INT/Engine/Blueprints/QuickStart/index.html
- Blueprint Editor Cheat Sheet
  - https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/CheatSheet/index.html

Debugging

- Window -> Developer Tools
  - Output Log, Message Log
- Blueprint Debugger
  - Set breakpoints
  - Watch variables
- Print String
  - to log output window
  - on screen
- Debug functions
  - `DrawDebug[…]`
    - Point, Line, Plane, Sphere…

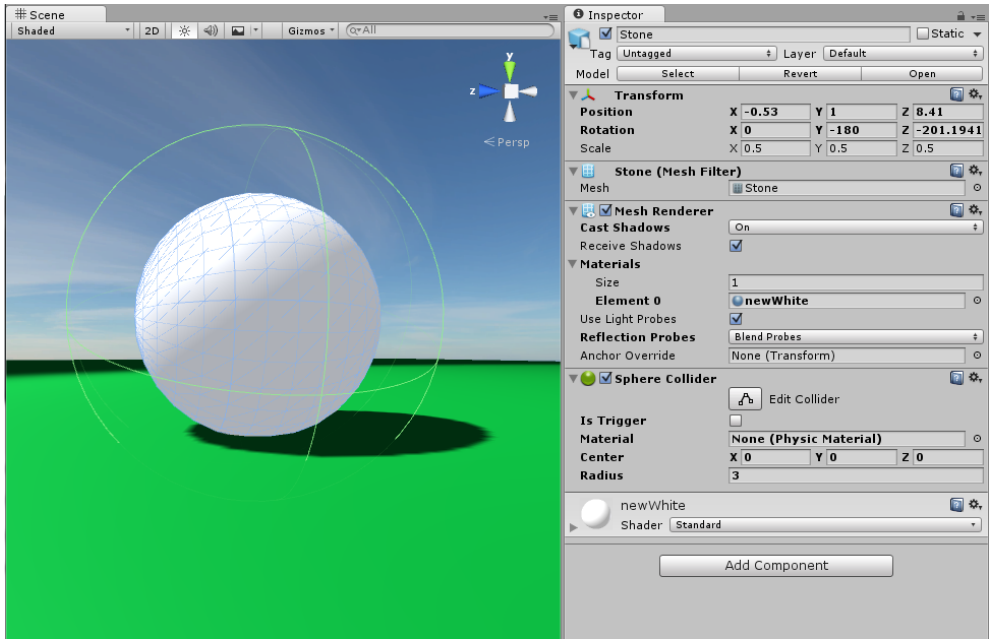




More on debugging Blueprints:
https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Debugging/

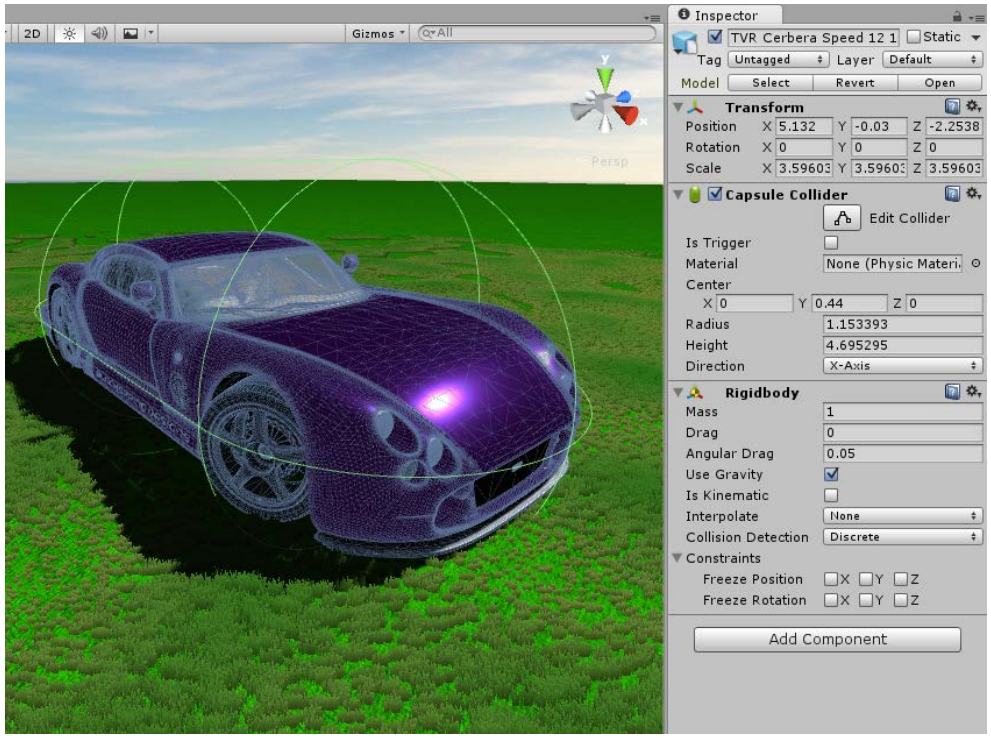
Basic Physics

Unity: Physics Basics



- Collider
  - Component that reacts on other Colliders
    - Different shapes, sizes and types
      - Primitive shapes
      - MeshCollider (not recommended)
    - At least one of the colliding Objects has to have a `Rigidbody` Component
    - `OnCollisionEnter()` event is called at the collision
  - Collider can be a trigger
    - `isTrigger` flag is set to true
    - Typically static
    - `OnTriggerEnter()` event is called at the collision
    - No physics is involved then

Unity: Physics Basics (2)



Rigidbody

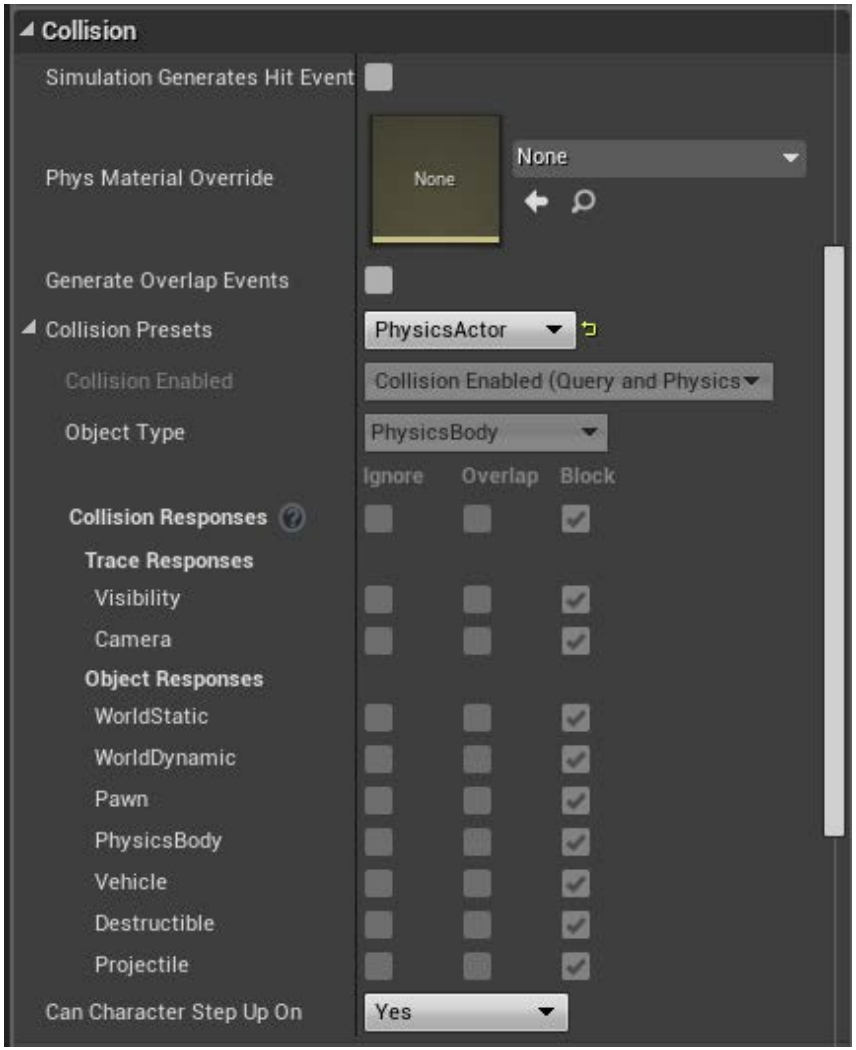
- Component that object to be effected by physics
- Must have a `Collider` Component to interact with other physics objects
- Enables gravity (castomisable)
- Define properties
  - Mass
  - Drag (air resistance)
  - Velocity
- `isKinematic` disables physics interaction for the object,
  - but still activates the colliders of other objects!

Unreal: Physics Basics (1)

- Physics settings are in Details
  - **Gravity** is set here
  - To make an object **simulate physics**
    - Simulate Physics flag = true
    - Needs at least a simple collision shape
    - Set mass of the object & damping factors
    - Have collision shapes
      - Simple: Primitives or convex hulls
      - Complex: Triangle mesh of the object
    - Mobility might be set to Movable
  - **Constraints**
    - lock position/rotation for individual axes

More on Physics:
https://docs.unrealengine.com/latest/INT/Engine/Physics/
https://docs.unrealengine.com/latest/INT/Resources/ContentExamples/Physics/index.html

Unreal: Physics Basics (2)



- Collision component is defined
  - Presets
  - Custom settings
- Collision must be enabled for all the objects involved
  - Block: e.g. solid objects
    - Enable **Simulation Generates Hit Events** flag to fire events on collision
  - Overlap: e.g. triggers
    - Enable **Generate Overlap Events** flag if you want them to fire (trigger colliders in Unity)
  - Ignore: no interaction
    - If you want two objects to block each other both of them need to be set to block the type of the other object

More on collision:
https://docs.unrealengine.com/latest/INT/Engine/Physics/Collision/Overview/index.html

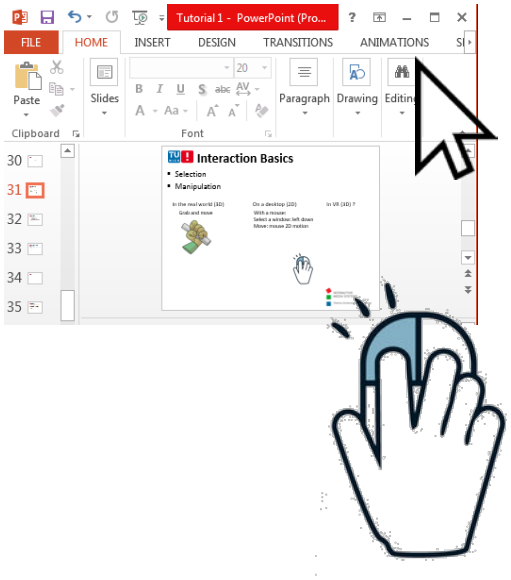
Interaction Basics

In the real world (3D)

Grab and move

On a desktop (2D)

With a mouse - 2D motion



In VR (3D) ?

Source: https://www.leapmotion.com

It‘s somewhere in-between

Leap Motion

Leap Motion Basics

- Tracks hands in close range: 10 – 60 cm

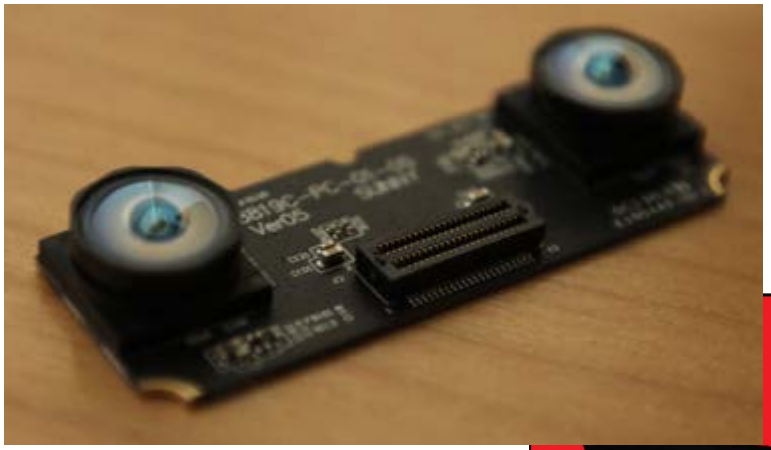
- Installation: Orion Beta 3.2
  - Unity in Asset store
    - Unity Core Asset
    - Add-on Leap Motion Interaction Engine
    - Optionally: Hands Module
  - Unreal Leap Motion Plugin

Source: https://www.leapmotion.com

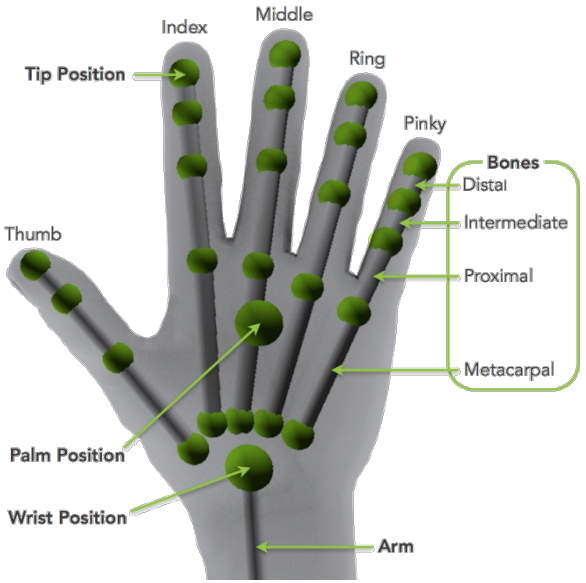
- Meant for a first-person view of the VE
  - best usability with an HMD
  - in VRUE use
    - unofficial human mounts
    - or desktop setup

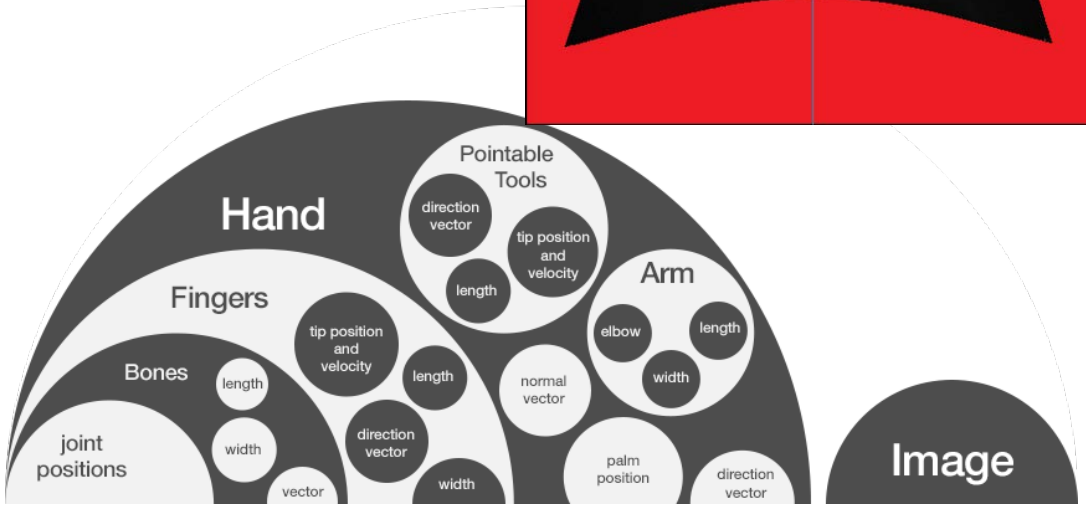
How to activate the plugin in UE4:
https://developer.leapmotion.com/unreal#103

How Does Leap Work



Source: https://www.ifixit.com



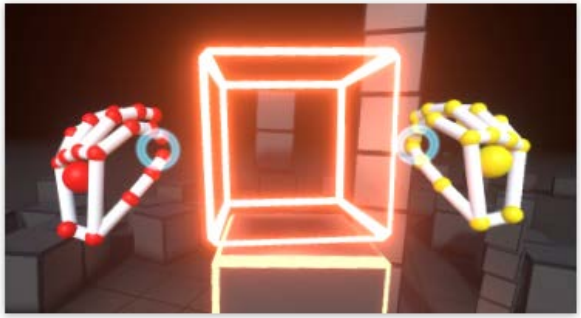


INTERACTIVE MEDIA SYSTEMS

Vienna University of Technology

Leap Motion Examples

Blocks (.exe only)

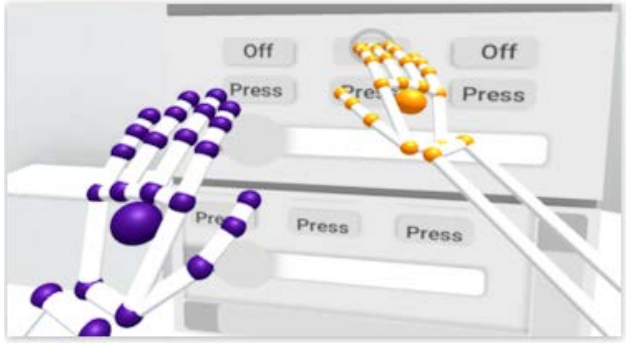


https://gallery.leapmotion.com/blocks

Unity Pinch Draw (Add-on module)

https://gallery.leapmotion.com/pinch-draw

Untiy UI Input Module (Add-on module)

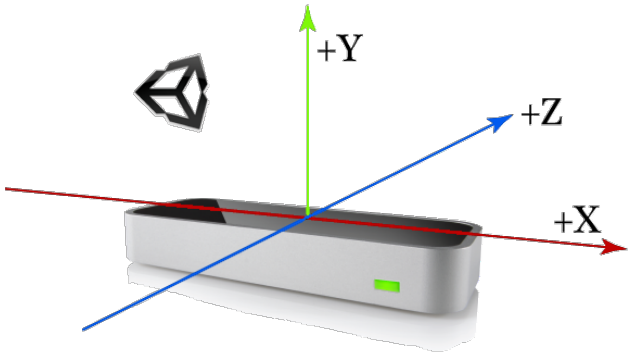


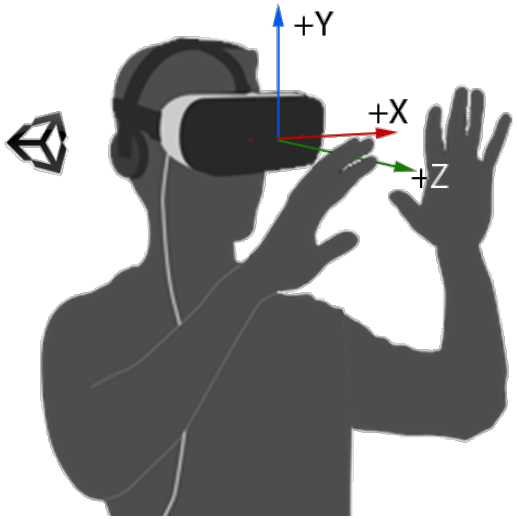
https://gallery.leapmotion.com/ui-input-module

Pictures source:
https://www.leapmotion.com

Unity: Leap Tracking Basics

- Position of a tracking target is
  - calculated in respect to the tracking origin in the real world
  - mapped to a position in the VE

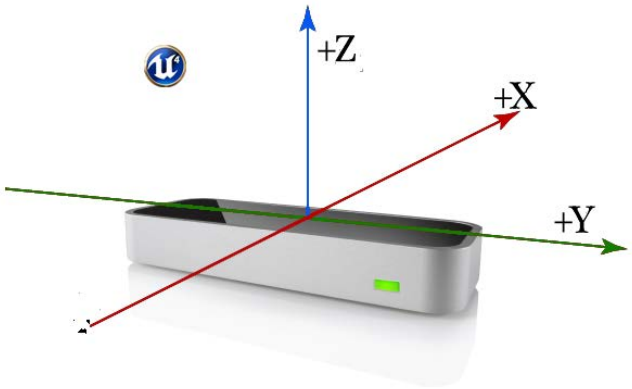


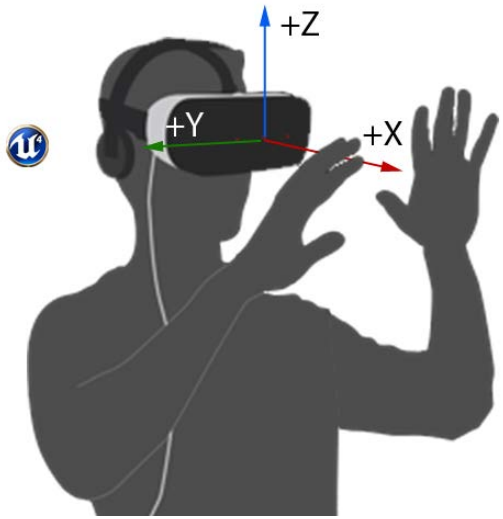


- Attention: Leap is sensitive to Light House IR lasers

Unreal: Leap Tracking Basics

- Position of a tracking target is
  - calculated in respect to the tracking origin in the real world
  - mapped to a position in the VE



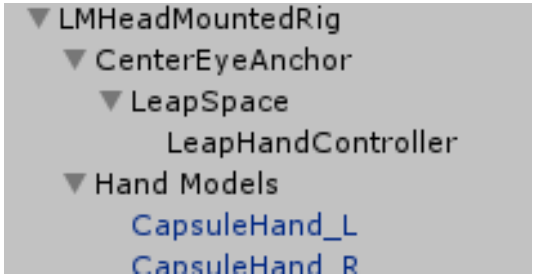


- Attention: Leap is sensitive to Light House IR lasers

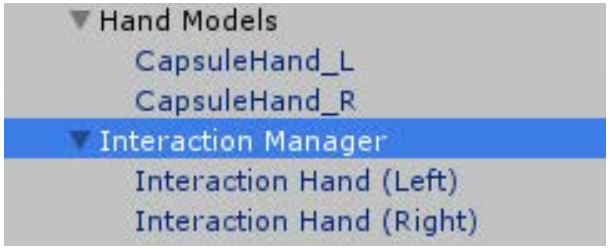
Image source and more information on coordinate systems:
https://developer.leapmotion.com/documentation/v2/unreal/devguide/Leap_Coordinate_Mapping.html

Unity: Integration

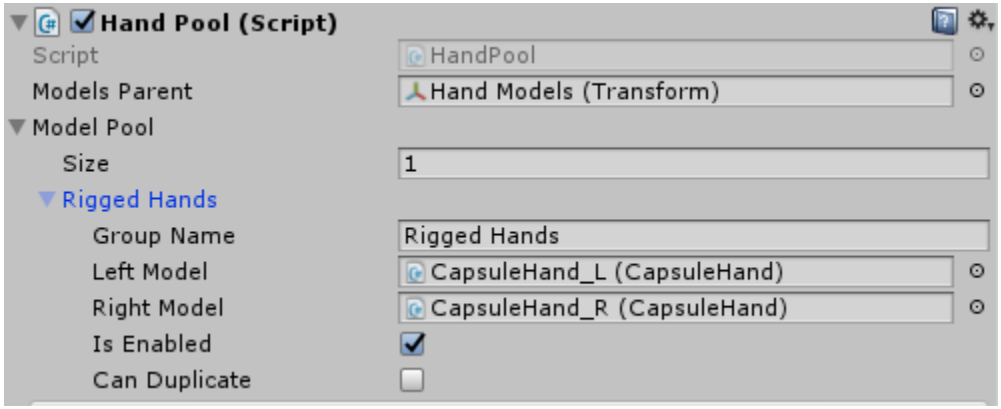
- Main Prefab



Extended for Interaction Manager (later)

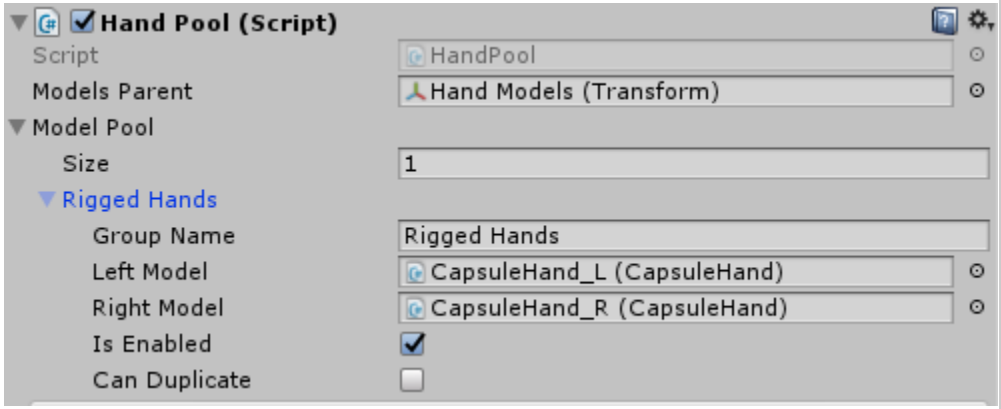


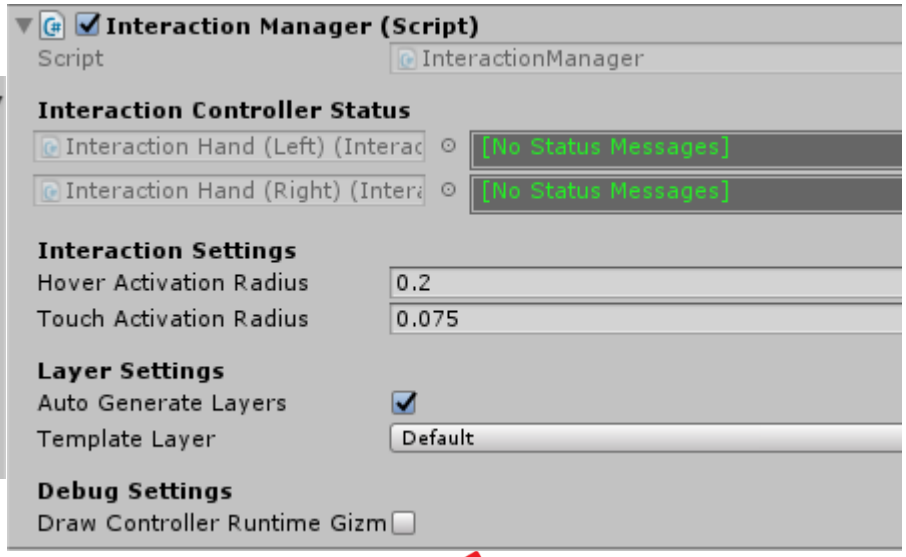
- LeapHandController contains main hand tracking – related scripts: LeapHandController, LeapServiceProvider and HandPool
- CenterEyeAnchor has a camera and HMD-related scripts (not so important in the task)
- HandPool defines hand models (make sure to drag correct game objects)



Unity: Natural Hand Interaction

- Interaction Engine (from Add-on modules)
  - allows to easier grab and throw objects
  - based on twisted physics rules (colliders, gravity, etc.)
- Usage
  - LMHeadMountedRigInteraction and corresponding `HandPool`
  - `InteractionManager`
  - `InteractionBehaviour` on game objects to be interacted with





Unity: Hand Types

- Capsule Hands
  - graphical representation, generated dynamically by a script
- Rigid Hands
  - made out of colliders, can be used to interact with game objects
  - are incompatible with Interaction Engine!
    - Remove from the hierarchy and the HandPool
- Interaction Hands
  - Invisible, used with Interaction Engine
  - Included to Leap Motion Interaction Engine v1.1 Add-on

Unity: Gestures with Leap

- Detector scripts (DetectionUtilities) for simple gestures:
  - Extended finger
  - Pinch
  - Finger direction
  - Palm direction, etc.
- Can be combined to create new gestures with `DetectorLogicGate`
  - For example, extended thumb and index make a pistol gesture

Pistol gesture

- Some gestures are recognized better than others:
  - In the plane orthogonal to the Leap z-axis
  - Neighbouring fingers are harder to distinguish
- Good examples in DetectionExamples (add-on module)

Unreal & Leap: Getting Started

- Leap Plugin is included in UE4, only needs to be enabled
- Choose a Blueprint Project, and the VR Template
- For information on the Leap Plugin see for example https://github.com/getnamo/leap-ue4
- The plugin contains blueprint characters you can place in the world
  - E.g. LeapRiggedCharacter, LeapFloatingHandsCharacter

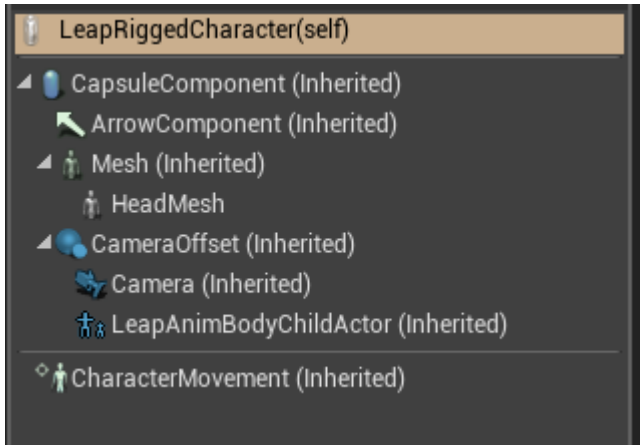
Unreal: LeapFloatingHandsChracter

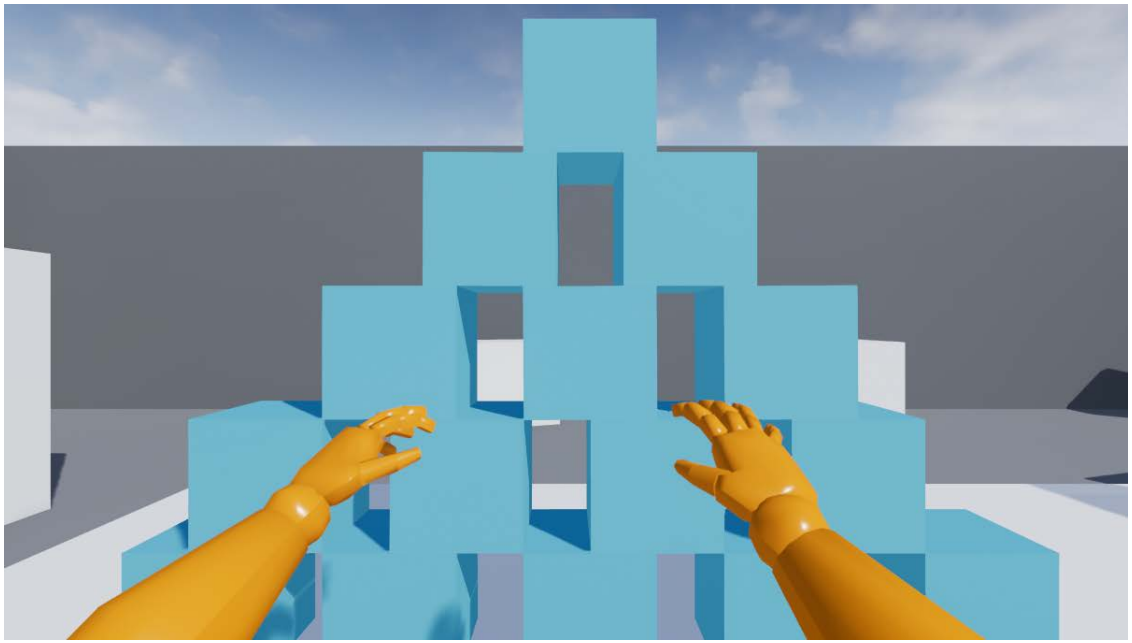
- Handtype can be changed at runtime
  - See the Event Graph of LeapFloatingHandsCharacter
  - Add the InputAction ChangeHands in the Project Settings

- Available hand types
  - LeapComponentHands
    - discrete components for the parts of the hands
  - LeapRiggedEchoHands
    - rigged mesh hand
  - LeapImageHands
    - displays images of your real hands from the camera sensors

Unreal: LeapRiggedCharacter

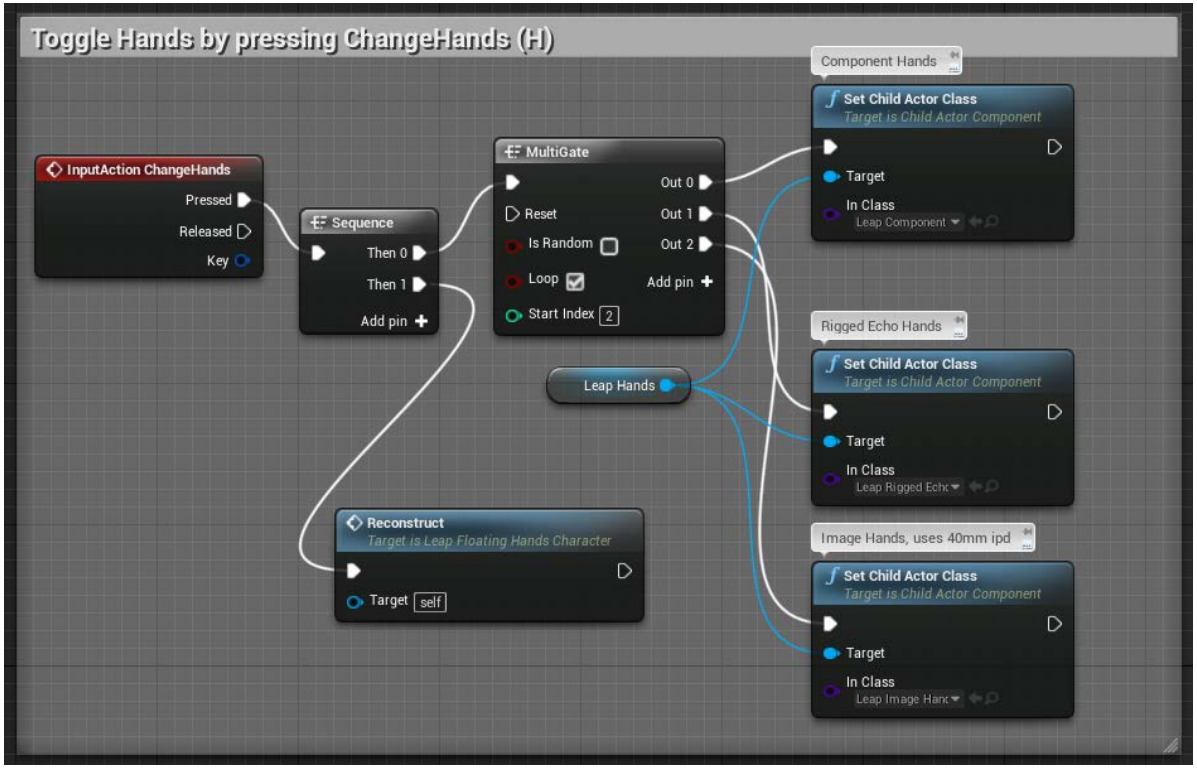
- Not just floating hands, full body





Unreal: Moving around

- Moving your character around via keyboard input
  - Look at the Movement Graph of `LeapBasicRiggedCharacter` or `LeapFloatingHandsCharacter`



Unreal: Gestures

- `LeapEventInterface`
  - Add Interface to a Blueprint
  - Add LeapController to that Blueprint

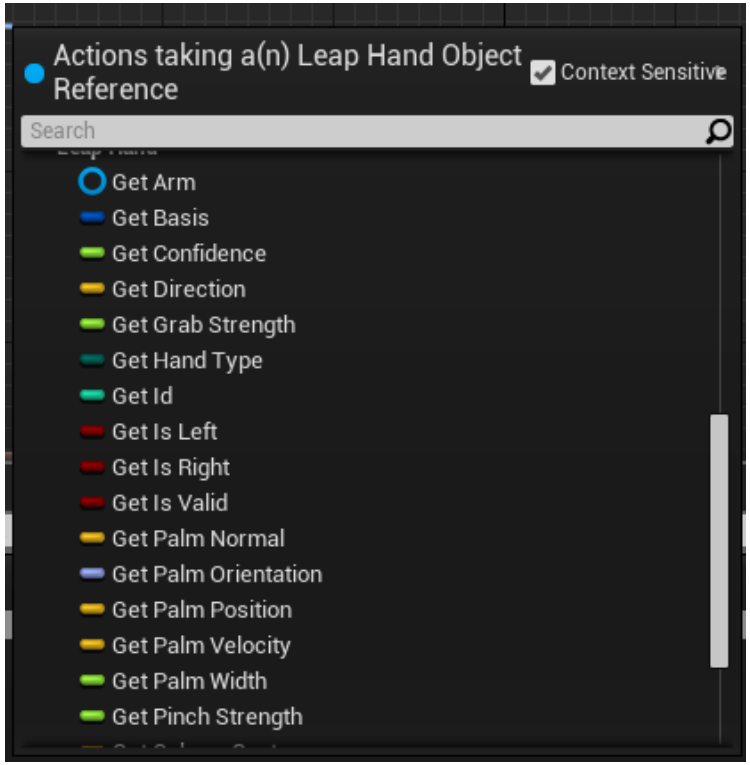
- The `LeapAnimBodyConnector` already implements that interface

- The Rigged Character and Floating Hands Character use the `LeapAnimBodyConnector`

- The LeapEventInterface
  - Events are for example **HandPinched** and **HandGrabbed**
  - Using the Event **HandMoved**, you can implement your own logic

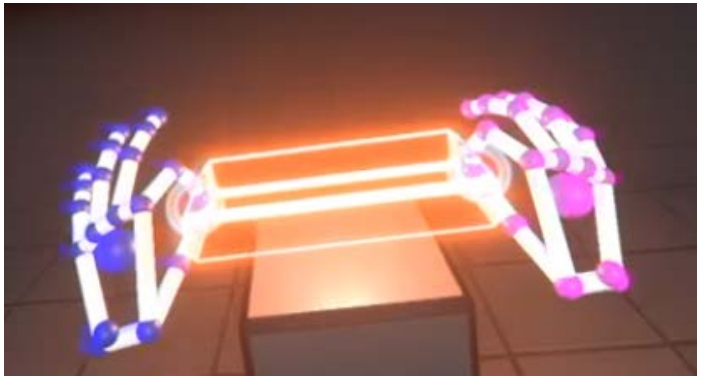
Unreal: Gestures

- **Events** of the `LeapEventInterface` return a Hand from which you can get a lot of information



3D Interaction

- 3D-Interaction
  - Different from 2D interaction
  - A user can move in all 3 dimentions
  - Tracking is the basis for the interaction



Source: https://www.leapmotion.com

- Basic principles
  - Real hand position and rotation is mapped to the Virtual Hand
  - In the Virtual Environment (VE), user interacts using Virtual Hands
  - Limitations:
    - LeapMotion tracking range
    - human grabbing distances

    } are too small to reach all the objects
  - Need interactions that are not possible in the real world
    - to reach and manipulate the distant objects

Task 1

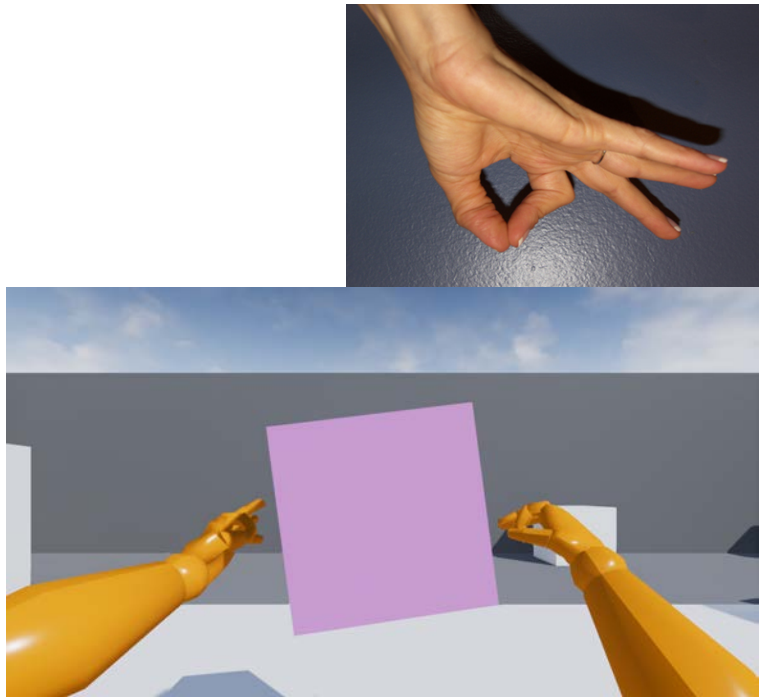
3D Interaction in Task 1

- 3D Interaction techniques
  - Looking at the scene from first-person perspective
  - Camera position in VE is fixed
  - Tracking origin – Leap Motion device

- Interaction techniques in VRUE
  - Interaction with VE
    - using gestures
  - Natural hand interaction with objects
    - Unity: using Leap Interaction Engine
    - Unreal: VR template & adding force
  - Distant interaction
    - using Raycast/LineTrace

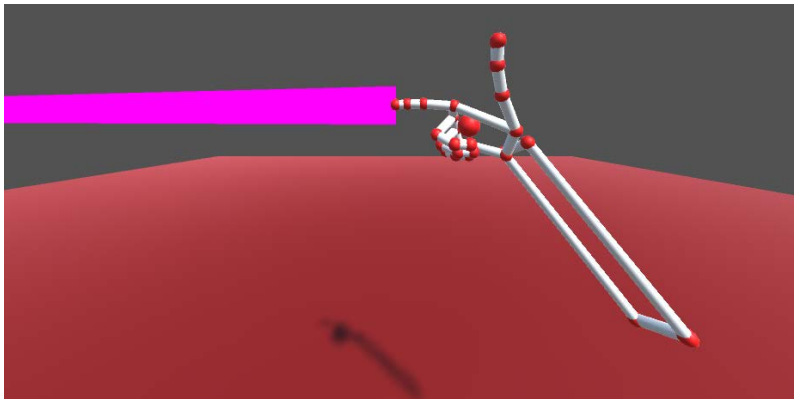
Create objects

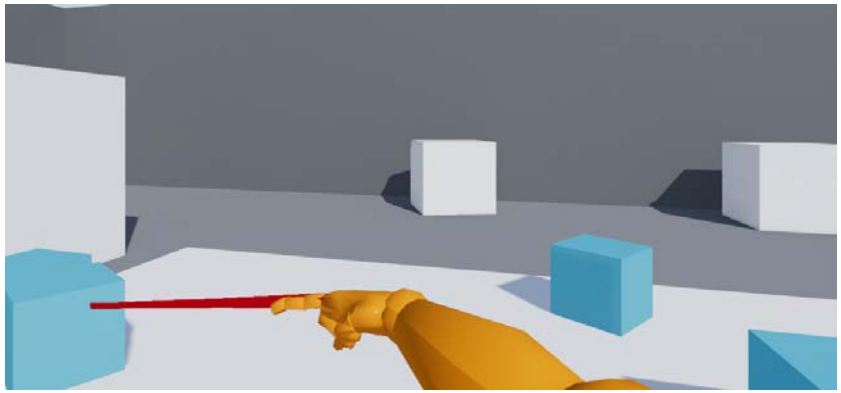
- With both hands doing Pinch
  - see Blocks demo
- Scale the object depending on the distance between left and right pinches
  - Unity:
    - Use `PinchDetector` and `DetectorGate`
    - Newly created objects should have `InteractionBehaviour` for Interaction Engine functionality
  - Unreal
    - `LeapEventInterface`
    - Events `HandPinched, HandGrabbed, HandMoved`



Select and Manipulate

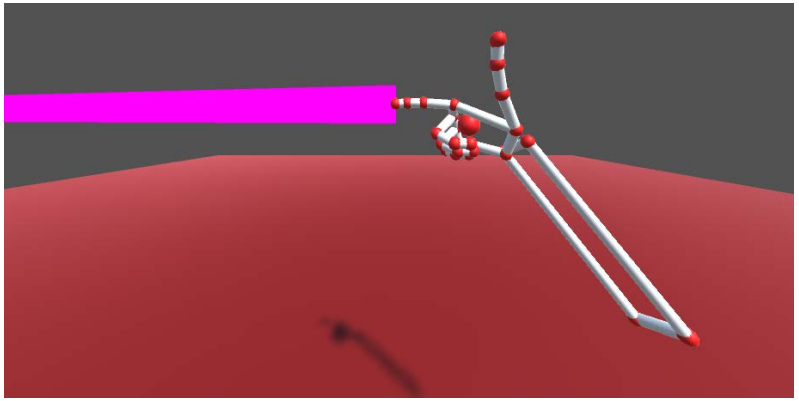
- In close range – reproducing real world hand interaction (Interaction Engine)
- In far range – using interaction metaphors (Raycast and gestures)





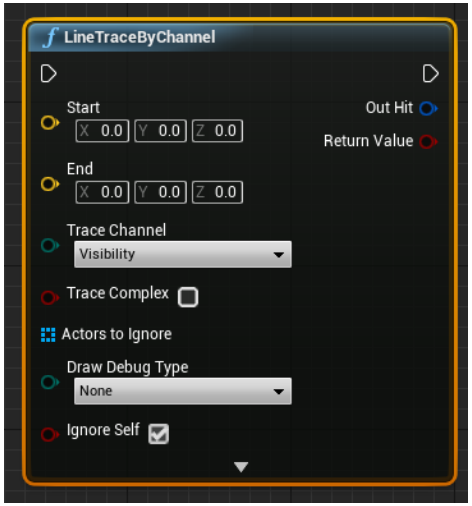
Unity: Raycast

- `Physics.Raycast`
  - Shoots an invisible ray
  - **True** when intersects with
    - Colliders
    - objects with Rigidbody, even without a collider
    - (if configured) Triggers
  - Collision info stored as `RaycastHit`
- Visualize with a `LineRenderer`

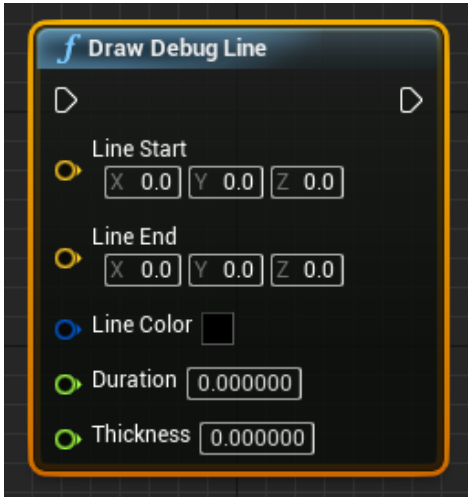


Unreal: TraceLine

- `LineTraceByChannel` or `LineTraceForObjects`
  - Shoots an invisible ray
  - Returns a `HitResult`



- Visualize by rendering a line
  - Simplest way: `Draw Debug Line`



More information on raycasts/traces:
https://docs.unrealengine.com/latest/INT/Gameplay/HowTo/UseRaycasts/Blueprints

Raycast/LineTrace Select

1. Cast a ray with Raycast/LineTrace when a pistol gesture is detected
2. An object currently hit by Raycast/LineTrace can be selected
3. Select by folding the thumb
4. Now the object can be manipulated

- Viasualize the selection state with the color
  - Color #1, when neutral
  - Color #2, when an object is hit by Raycast
    - but not yet selected
  - Color #3, when selected and can be manipulated

Task 1 – Assignment Submission

- Assignments submissions are held in TUWEL exclusively!

- You submit:
  - .zip of the Unity project
    - Include “feature description” in a txt file for bonus
      - A list of what is done
  - Zip-file might be up to 256MB

- Deadline: 15.10.17, 23:59
  - Every extra day = -10% of your points

- Grading: main points + bonus points

Grading

- Please, read the assignments carefully
  - Project organization is important and graded!
  - Scene organization is also graded!
  - Clean code/blueprints are appreciated
    - Comments are very welcome
  - Cheating will be punished!

- Bonus is optional
  - But the earned points are counted into the grade, if something was not perfect
  - Doesn’t mean you can skip the main functionality

Unity: Where to Look for Help

Official Tutorials	https://unity3d.com/learn/tutorials
Other Video Tutorials	http://www.unity3dstudent.com/category/modules/
User Manual	http://unity3d.com/support/documentation/
Reference Manual	https://docs.unity3d.com/432/Documentation/Components/
Scripting Reference	https://docs.unity3d.com/ScriptReference/index.html
Community Forum	https://forum.unity3d.com/
Knowledge Base	https://support.unity3d.com/hc/en-us

Leap Motion (Orion)

https://developer.leapmotion.com/documentation/unity/index.html

https://github.com/leapmotion/UnityModules/wiki/Getting-Started-%28Interaction-Engine%29

Unreal: Where to Look for Help

Official Tutorials	https://docs.unrealengine.com/latest/INT/
Video Tutorials	https://docs.unrealengine.com/latest/INT/Videos
Community Forum	https://wiki.unrealengine.com/Main_Page
Unity to Unreal	https://docs.unrealengine.com/latest/INT/GettingStarted/FromUnity
Blueprints	https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html
Content Examples	https://docs.unrealengine.com/latest/INT/Resources/ContentExamples
Leap UE4 Docs	https://developer.leapmotion.com/documentation/v2/unreal/index.html
	https://github.com/getnamo/leap-ue4

Questions?