Unreal Engine and Blueprints

# Overview

The purpose of this document is to cover the various important concepts and elements of the Unreal Engine and its Blueprint Node System.

# Actors

One of the most fundamental elements of the Unreal Engine. Actors have their own event graphs for handling blueprint scripting logic. Actors are objects that exist in our scene. Actors themselves have the default functionality that every object in a game scene would need:

* A position, rotation and scale. (Transform)
* A Root Component
* Default Functions for Begin play, tick and construction

The actors themselves have additional logic on top of them in the form of components. These can be a variety of types, common ones including:

* Static Mesh
* Skeletal Mesh
* Camera
* Spring Arm
* Character Movement

Using inheritance, you can create a child of the actor class to suit your games requirements. Important child actors to be aware of are:

* GameMode
* Controller
* Pawn
* Character

Creating an Actor:

Right click in your content browser and choose create blueprint class.

A screenshot of a computer

AI-generated content may be incorrect.

## GameMode

Main area for your games current logic. You can have a default game mode set which will always be used with every map unless overridden. Override means you are choosing a different game mode specific to the current map you are in.

Game modes will have details on the default pawn class you use (optional), the controller, game state and player state.

A screenshot of a computer

AI-generated content may be incorrect.

## Level Blueprint

EventGraph that is specific to the current level you are in. Because it is specific to the current level you will be able to refer to actors in the scene through this blueprint. To do this just select and object in your scene, open the level blueprint, right click on the event graph and you can create a direct reference there.

Try to avoid having all your game logic here as this is only specific to the level you are currently at. Keep the script specific to the level itself here.

## Pawn

An object that can be possessed by a controller. Default is an AI controller, but a player controller can possess the pawn and use physical input commands to tell the pawn what to do.

## Character

Extension on the Pawn class, this takes the idea of a pawn and takes it further with a default skeletal mesh component, character movement component for handling most movement logic a character would need and a capsule collider. There are a lot of ways to tell the character how to move/crouch/jump/etc.

# Scripting

Unreal Engine uses a system called blueprints for handling its code in editor. This is done on an event graph where we can place nodes and connect them.

A screenshot of a computer

AI-generated content may be incorrect.

* Brown: Components attached to this current actor.
* Purple: Tabs are currently open for this actor/event graphs. Press the middle mouse button to close them.
* Orange: All code information, eventgraphs, functions, macros, variables and event dispatchers.
* Red: Event Graph Viewport.
* Green: Details on currently selected element.

## Nodes

The bread and butter when it comes to visual scripting. Nodes come in a variety of shapes and forms, from variables, to functions, events, etc.

Let’s break down how nodes are made.

A screen shot of a computer

AI-generated content may be incorrect.

* Blue: Event node, displayed with a red header.
* Green: Execution pin, the direction of how logic flows from one point to another, will need to find a next execution pin to move to should logic need to continue.
* Red: Variable node contains information about whatever value this variable is.
* Purple: Input node, in this context it is an integer value that is taking in the value of Variable.
* Orange: Function node, blue header, has specific logic ideally tied to the name of the function, double click to enter this function node’s logic to see what it does.

Variable types are color coded to make things easier to read/understand. Blue being objects.

A screenshot of a phone

AI-generated content may be incorrect.

Actors and parent classes normally have override functions attached to them you can use in the child. BeginPlay and Tick are such examples, you can find them in the override section on the actor itself.

A screenshot of a computer

AI-generated content may be incorrect.

## Node Details

* Instance Editable: Allows you to change the value on the actual instance. Helpful for having a dynamic object that can have different specifics within the level itself.
  + 
* Pure: removes execution pins from a function. Helpful for functions that don’t require order.
  + A screenshot of a computer

    AI-generated content may be incorrect.
* Inputs and Outputs: Parameters and return values from a function.
  + A screenshot of a black screen

    AI-generated content may be incorrect.
* Show 3D widget: Helpful for a Vector3, will display the particular widget on the instance within the level.
  + A screenshot of a computer

    AI-generated content may be incorrect.
  + A screenshot of a video game

    AI-generated content may be incorrect.

## Data Structures

A screenshot of a computer

AI-generated content may be incorrect.

* Single Data Type will just be one value of that particular type.
* Array is a list of values.
* Set is a list of unique values, cannot accept duplicates.
* Map is a key value list, one key and one value. Helpful for many different reasons!
  + Pass in a key to return the value. Key and value can be any variable type.

# Node Library

|  |  |
| --- | --- |
|  | Fired off when the game begins or when the actor is created during gameplay. |
|  | Event that fires off every single frame. Delta time is the time since the last frame. |
|  | Fired off when the object is created. |
|  | Print string will print a message to the screen and the console log. For additional details, the key is the line that it will be printed on, helpful for when you don’t want your debug statements cascading downward. |
| A screenshot of a computer  AI-generated content may be incorrect. | Branch node, the if statement for handling logic. Requires a Boolean to determine if true or false. |
|  | Sequence node will carry out logic in a specific order. Add as many pins as you need. |
|  | Cast to class of your choice, good at checking to see if the object is of a specific class type. Will have execution if successful or fails. Finally, will output the correct object as that class if successful. |
|  | Spawn actor from class, will create a specific actor class at a location of your choosing. |
|  | Get Actor (or Actors) of class will return the first actor it can find (or an array if Actors). |
|  | Delay Node, will pause execution for a set amount of time. Must be created in the eventgraph, will not work in functions. |
|  | Will cause the delay to refresh its cooldown if called again. |
|  | Set timer by event or string, will cause a particular event or function to execute after a specified amount of time. Can loop. Also return a reference to the timer should you need to change it while waiting. |
|  | Timeline node, good at handling tick functionality for a specific track of time. Double click to enter the timeline to set a track on it. Will return a value based on a curve track. |
|  | Switch statement, will execute line of code based on the input selection. |
|  | Will destroy specified actor. |
|  | Will destroy specified actor after a set amount of time. |
|  | Flip flop will first execute A when called the first time, then switch to B when called a second, will continue to switch between the two every time it has been called. |

# Programming References

## Classes

* Blueprints for a particular object, you can create the object (called an instance) via spawn actor.
* You can create child classes of these classes (most objects in Unreal inherit from the Actor class).
  + This will inherit all parent functionality.
* You can see all of the override functions from the class on the functions section.
  + A screenshot of a computer

    AI-generated content may be incorrect.

## Structure

* Data containers for variables.
* Helpful when you want to have a whole bunch of related variables in one space.
* Also helpful for re-using.

## Interface

* A contract that can have a set of functions on it.
* When attached to a class it will guarantee that the function exists on it.
* You can then cast to that interface to call the function on the specific class.
* Example: You are swinging your sword and hit something. Instead of checking all possible things you could hit you could instead check if it has the iHittable interface on it and call the on hit function iHittable has.

## Enumeration

* Numerical representation of a string.
* Helpful for when you want to compare to a number (if current stance == 1). It will give more context and meaning.
* Also helpful at restricting what values the programmer can use for this data type.
  + Example: Setting movement speeds via enumeration so you can’t accidently set a different value unintentionally.

## Macros

* C++ concept
* Replace a defined node with a block of code.
* This can be helpful when you need multiple execution nodes going in and out of chunk of code.

# Enhanced Input

Enhanced input is a nice way to organize your input into different contexts based on the current situation your player is experiencing. You can layer on multiple contexts as well with priority orders on which inputs take precedence.

To make this work you need to make an input mapping context along with an input action for each potential input you are expecting your player to use.

A screenshot of a computer

AI-generated content may be incorrect.

Ensure the input action is specifying the correct data return type (bool, pressed or not, axis which returns a number between -1 and 1).

Assign the correct input in the input mapping context.

Make sure to grab the enhanced input local subsystem and assign it the correct context.

A screenshot of a computer

AI-generated content may be incorrect.

Then you can call the actions on your character.

A screenshot of a computer

AI-generated content may be incorrect.

Note: For older projects that were using the older input system, you will need to enable the plugin and set the input classes correctly to the enhanced input system.

A black and white striped background

AI-generated content may be incorrect.

Located in project settings -> Input

# Packaging the Project

* Make sure to set your game default map to the correct one.
  + -> Project Settings, Maps & Modes
  + A black rectangular object with a white border

    AI-generated content may be incorrect.
* Make sure to specify the exact maps you want to be using throughout your project. I find the best way to find this section is to search “maps” in project settings.
  + A screenshot of a video

    AI-generated content may be incorrect.
* Choose the platform and select build.
  + A screenshot of a computer

    AI-generated content may be incorrect.
* It may take some time for the project to build!

# Debugging

Console Logging: Print String

A screenshot of a computer

AI-generated content may be incorrect.

* Drawing various shapes can be helpful for visualizing what is happening in our code.
* Breakpoints
  + Press F9 on a node to create a breakpoint, code execution will stop here.
  + A screen shot of a video game

    AI-generated content may be incorrect.
  + A close-up of a screen

    AI-generated content may be incorrect.
    - Green: Navigate to current node that execution is at.
    - Red: Stop debugging and resume game execution.
    - Purple: Step into a node (or function) to view execution within.
    - Blue: Next step
    - Yellow: Step out of the current node you are in

# Useful Tips

* Dragging off variable type will give you an option to create a variable.
  + A screenshot of a computer

    AI-generated content may be incorrect.
* CTRL + Space to open the content browser.
* ALT + Left Click to remove a connection between nodes.
* Select multiple nodes and press Q to straighten the lines.
* You can select the default editor startup map in Project Settings -> Maps & Modes.
  + This will ensure you go straight to this map whenever you open your editor.
* You can filter the content browser, helpful for navigating your assets quicker.
  + All assets are color coded! Be mindful of this. (example static meshes are cyan color)
* Collections can be a nice way to put assets from your content browser into one section.
  + This is helpful for not needing to browse through large amounts of assets constantly when you are wanting to just need specific ones.
* Naming Conventions!
  + Generally prefix blueprints with the relevant object name.
  + BP\_ for basic blueprints
  + ABP\_ for animation blueprints
  + WBP\_ for widget blueprints
  + SK\_ for skeletal meshes
  + M\_ for materials
  + Etc
  + Booleans should be prefixed with a lower case b.
* If things are getting too busy on an event graph try to consolidate your blocks of code in functions and macros.