Unreal Intro Project: Checkers :D

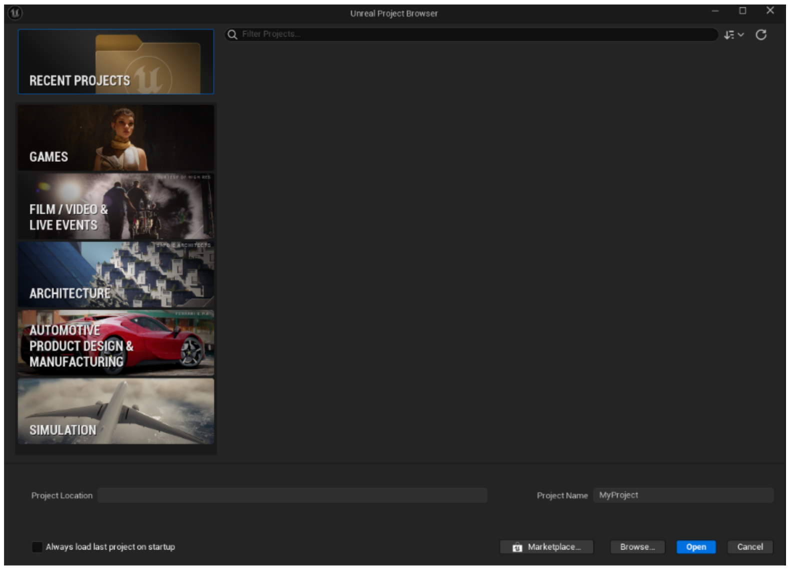
By LoganHall195

Unreal Editor
Unreal Editor 5.4.0
35% - Compiling Global Shaders..
Copyright © Epic Games, Inc. All rights reserved.

PDF v1.0

Creating Your Project

After launching Unreal Engine for the first time, you will be met with this screen:



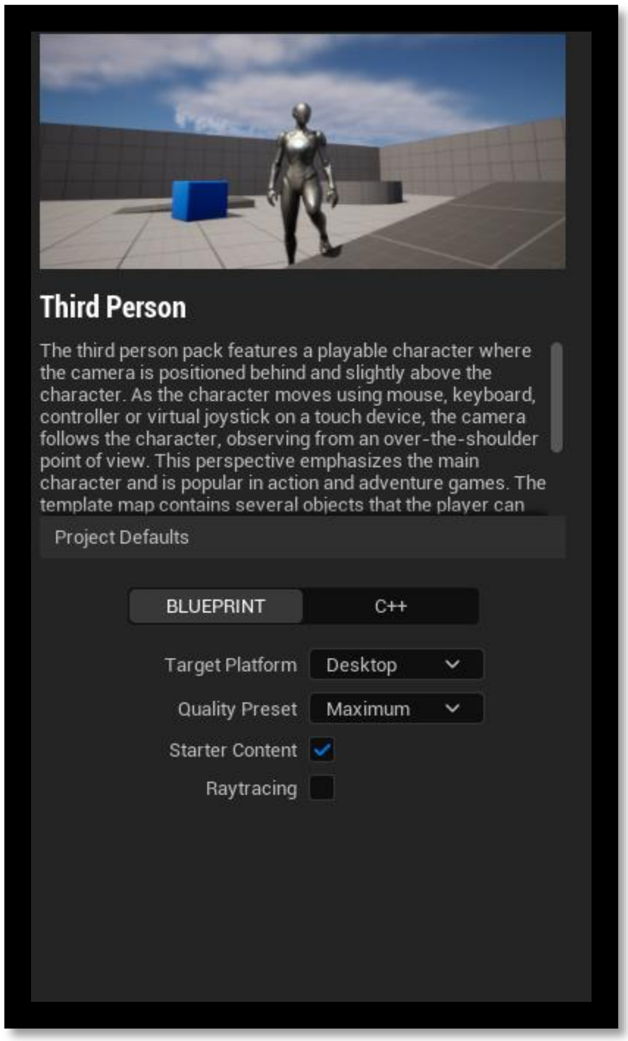
From here you will do a bunch of important things to set the stage for your project, these include:

- Choosing the a base template for your project. Making a car game? Start with a "Game: Vehicle" template!
- Choosing a directory for your project, since you will likely be keeping all of your projects in the same place, you will want to make sure wherever you choose has a LOT of space. At least ~20 gb.
- Naming your project. Your project name can be changed, but... Let's face it, the name is important.

"For all of my projects, I like to use the 'Game: Third Person' template with 'Starter Content' enabled. This gives me the opportunity to go in any direction with my game. Even if I want to change to my game mode to a different POV or style later!" – Joe Rogan

---

It is recommended to start with a "Game: Third Person" template for this project. Make sure you enable "Starter Content" as well!



Third Person

The third person pack features a playable character where the camera is positioned behind and slightly above the character. As the character moves using mouse, keyboard, controller or virtual joystick on a touch device, the camera follows the character, observing from an over-the-shoulder point of view. This perspective emphasizes the main character and is popular in action and adventure games. The template map contains several objects that the player can

Project Defaults

BLUEPRINT | C++

Target Platform: Desktop

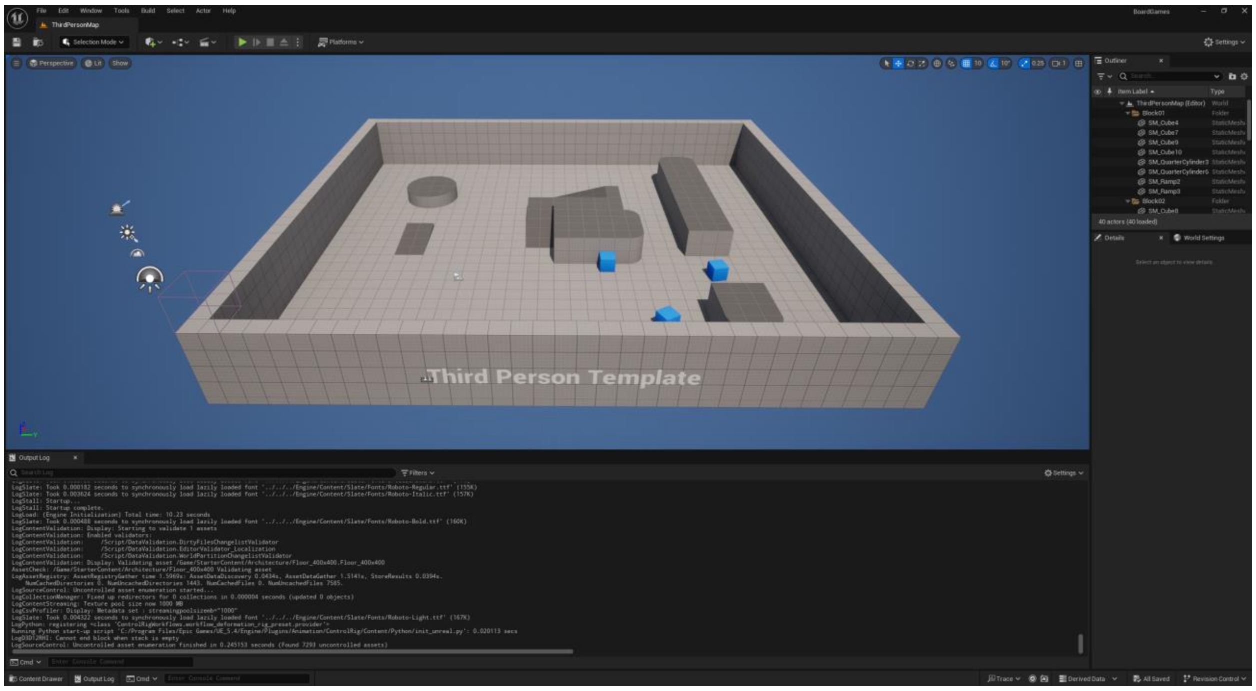
Quality Preset: Maximum

Starter Content: ✓

Raytracing: ☐

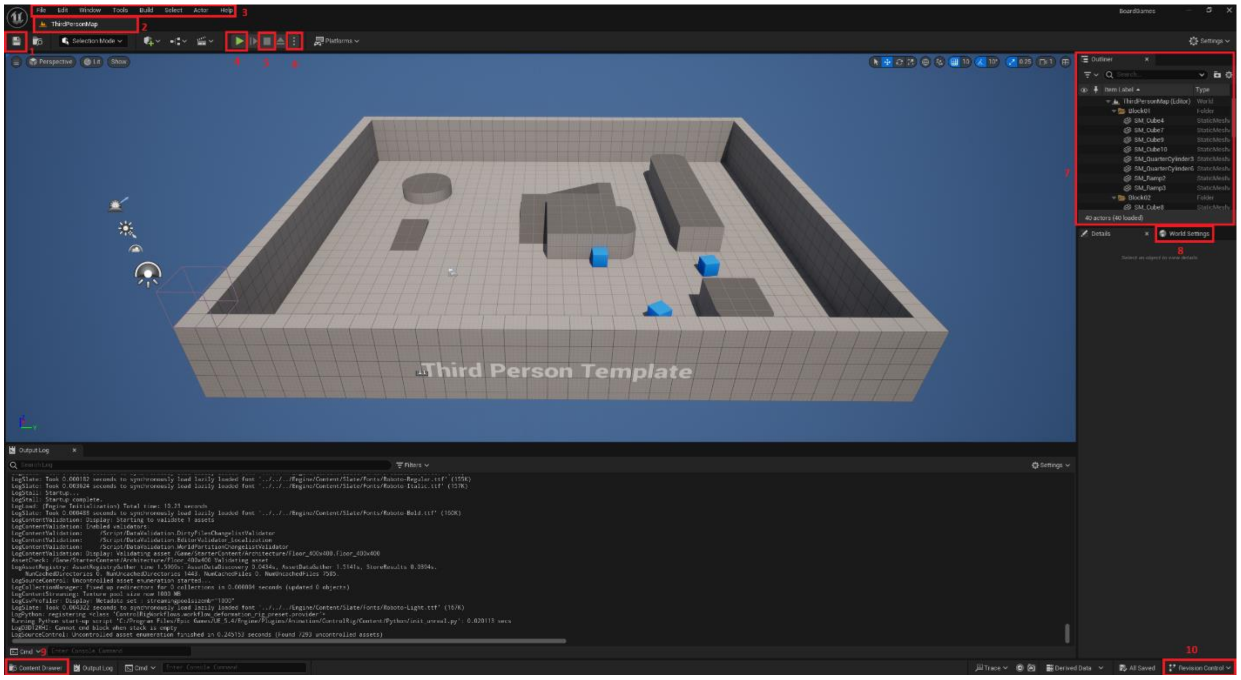
Name this project whatever you like, if you want, you can just leave it “MyProject”.

This is likely what you will see when you open up your project for the first time.



```
LogSlate: Took 0.000182 seconds to synchronously load lazily loaded font '../../../Engine/Content/Slate/Fonts/Roboto-Regular.ttf' (155K)
LogSlate: Took 0.003624 seconds to synchronously load lazily loaded font '../../../Engine/Content/Slate/Fonts/Roboto-Italic.ttf' (157K)
LogStall: Startup...
LogStall: Startup complete.
LogLoad: (Engine Initialization) Total time: 10.23 seconds
LogSlate: Took 0.000488 seconds to synchronously load lazily loaded font '../../../Engine/Content/Slate/Fonts/Roboto-Bold.ttf' (160K)
LogContentValidation: Display: Starting to validate 1 assets
LogContentValidation: Enabled validators:
LogContentValidation:     /Script/DataValidation.DirtyFilesChangelistValidator
LogContentValidation:     /Script/DataValidation.EditorValidator_Localization
LogContentValidation:     /Script/DataValidation.WorldPartitionChangelistValidator
LogContentValidation: Display: Validating asset /Game/StarterContent/Architecture/Floor_400x400.Floor_400x400
AssetCheck: /Game/StarterContent/Architecture/Floor_400x400 Validating asset
LogAssetRegistry: AssetRegistryGather time 1.5969s: AssetDataDiscovery 0.0434s, AssetDataGather 1.5141s, StoreResults 0.0394s.
    NumCachedDirectories 0. NumUncachedDirectories 1443. NumCachedFiles 0. NumUncachedFiles 7585.
LogSourceControl: Uncontrolled asset enumeration started...
LogCollectionManager: Fixed up redirectors for 0 collections in 0.000004 seconds (updated 0 objects)
LogContentStreaming: Texture pool size now 1000 MB
LogCsvProfiler: Display: Metadata set : streamingpoolsizemb="1000"
LogSlate: Took 0.004322 seconds to synchronously load lazily loaded font '../../../Engine/Content/Slate/Fonts/Roboto-Light.ttf' (167K)
LogPython: registering <class 'ControlRigWorkflows.workflow_deformation_rig_preset.provider'>
Running Python start-up script 'C:/Program Files/Epic Games/UE_5.4/Engine/Plugins/Animation/ControlRig/Content/Python/init_unreal.py': 0.020113 secs
LogD3D12RHI: Cannot end block when stack is empty
LogSourceControl: Uncontrolled asset enumeration finished in 0.245153 seconds (Found 7293 uncontrolled assets)
```

Lets take stock of what is on the screen...

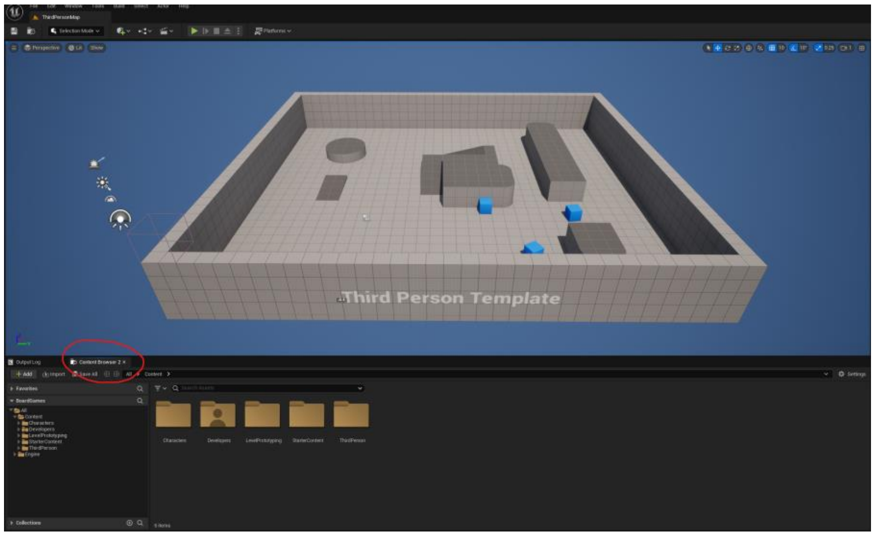


```
LogSlate: Took 0.000182 seconds to synchronously load lazily loaded font '../../../Engine/Content/Slate/Fonts/Roboto-Regular.ttf' (155K)
LogSlate: Took 0.003624 seconds to synchronously load lazily loaded font '../../../Engine/Content/Slate/Fonts/Roboto-Italic.ttf' (157K)
LogStall: Startup...
LogStall: Startup complete.
LogLoad: (Engine Initialization) Total time: 10.23 seconds
LogSlate: Took 0.000488 seconds to synchronously load lazily loaded font '../../../Engine/Content/Slate/Fonts/Roboto-Bold.ttf' (160K)
LogContentValidation: Display: Starting to validate 1 assets
LogContentValidation: Enabled validators:
LogContentValidation:     /Script/DataValidation.DirtyFilesChangelistValidator
LogContentValidation:     /Script/DataValidation.EditorValidator_Localization
LogContentValidation:     /Script/DataValidation.WorldPartitionChangelistValidator
LogContentValidation: Display: Validating asset /Game/StarterContent/Architecture/Floor_400x400.Floor_400x400
AssetCheck: /Game/StarterContent/Architecture/Floor_400x400 Validating asset
LogAssetRegistry: AssetRegistryGather time 1.5969s: AssetDataDiscovery 0.0434s, AssetDataGather 1.5141s, StoreResults 0.0394s.
    NumCachedDirectories 0. NumUncachedDirectories 1443. NumCachedFiles 0. NumUncachedFiles 7585.
LogSourceControl: Uncontrolled asset enumeration started...
LogCollectionManager: Fixed up redirectors for 0 collections in 0.000004 seconds (updated 0 objects)
LogContentStreaming: Texture pool size now 1000 MB
LogCsvProfiler: Display: Metadata set : streamingpoolsizemb="1000"
LogSlate: Took 0.004322 seconds to synchronously load lazily loaded font '../../../Engine/Content/Slate/Fonts/Roboto-Light.ttf' (167K)
LogPython: registering <class 'ControlRigWorkflows.workflow_deformation_rig_preset.provider'>
Running Python start-up script 'C:/Program Files/Epic Games/UE_5.4/Engine/Plugins/Animation/ControlRig/Content/Python/init_unreal.py': 0.020113 secs
LogD3D12RHI: Cannot end block when stack is empty
LogSourceControl: Uncontrolled asset enumeration finished in 0.245153 seconds (Found 7293 uncontrolled assets)
```

1. Save Button	**2.** Level/Map	**3.** Window Bar	**4.** Play Button	**5.** Stop Button
6. Play Settings	**7.** Level Outline	**8.** World Settings	**9.** Content Drawer	**10.** Version Ctrl

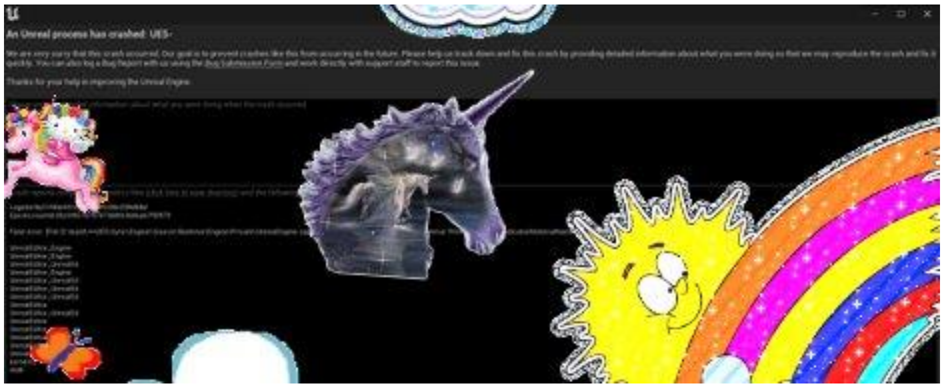
Content Drawer

1. Open up your Content Drawer (Bottom-Left)
2. Dock in Layout (Top-Right)
    a. You may need to drag it to the correct location if it docks incorrectly.



3. Take notice of file structure
    a. Your "Content" folder is your project.
    b. By default, "Characters" contains the UE4 and UE5 mannequins as well as their default animations.
    c. "Developers" folder will have files that you want to use but that other developers will not need. Examples include WIP items that are broken but that you want to be backed up when you push them to your branch. However you generally wont be using this folder.
    d. "Level Prototyping" by default, contains the starter items that are provided for the world you load into. For larger projects you may want to change this. Especially if you are creating your own level and ditching the starter one.
    e. "Starter Content" has a bunch of stuff that you should really check out yourself. It includes props, example items such as lamp blueprints, maps, and textures!
    f. "Third Person" contains your template items which in this case includes your BP_ThirdPerson_Character (Player Character), your IMC_Default (input mappings), and your ThirdPersonMap (Level that you see loaded in front of you).

4. Lets move on to setting up our environment...
    a. Create your own folder under content and name it "_Main". The underscore ensures that it stays at the top of the content browser for ease of access.
    b. Inside the "_Main" folder, create the following folders:
        i. Blueprints
            1. GameBoards
            2. Props
        ii. Characters
            1. Mixamo
        iii. Misc
5. Finally lets take a few more notes about the Content Browser before moving on...
    a. By right clicking on an item, you can select "Show in explorer" to see the actual file location of items.
    b. Dropping stuff into the file explorer does not automatically bring them into your project where they can be used.
        i. To import items, click the import button in the top left of your content browser, navigate to the item, and then select the item. You can not import assets from other projects. You can only migrate them (or export/import).
    c. Deleting referenced items in file explorer is generally bad, especially if they are referenced elsewhere in your project.
        i. While UE5 generally does a really good job at letting you delete referenced items, it is always best practice to check yourself before deleting things as it ~could~ corrupt your project :D



An Unreal process has crashed: UE5-

We are very sorry that this crash occurred. Our goal is to prevent crashes like this from occurring in the future. Please help us track down and fix this crash by providing detailed information about what you were doing so that we may reproduce the crash and fix it quickly. You can also log a Bug Report with us using the Bug Submission Form and work directly with support staff to report this issue.

Thanks for your help in improving the Unreal Engine.

Creating Your First Blueprint

What is a blueprint anyways?

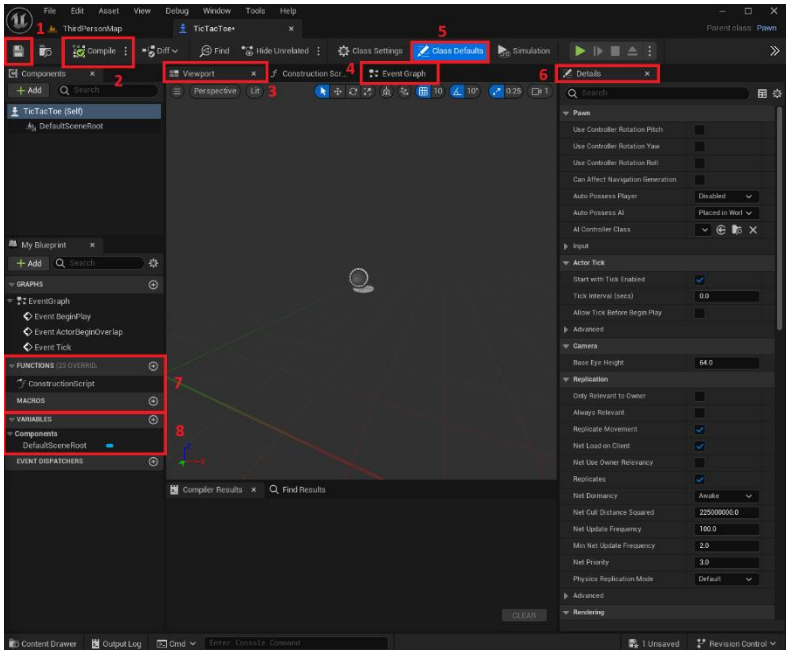
- A blueprint is a collection of components (static meshes, collision boxes, character movement components, etc.) that when placed into a world, are more than just a simple mesh.
- So if you put a simple rock into the world, it would probably be a mesh. But if you wanted to make it so you could mine that rock with a pickaxe, you would want to make it into a blueprint.

Instructions:

1. Navigate to "_Main > Blueprints > Gameboards" and right click inside the folder
2. Select "Blueprint Class"
    a. Take note of the options here, lets dive a little deeper into them.
        i. Actor: Lets say you have a "Door" mesh(3D Model) you want it to be placed around the level , with the ability to be opened or closed by the player. You would create an Actor Class "Door"
        ii. Pawn: Is the recommended Class for a "Playable Actor" , controlled by the Player or AI.
        iii. Character: A pawn that has legs and can walk. Think "BP_Third_Person_Character".
        iv. Actor Component: A component is something that modifies all actors that have this component. So if you wanted to have multiple races for characters in a fantasy RPG, you may want to give them all a component to manage their inventory.
3. Select "Pawn"
4. Name: TicTacToe

Now when you open the new blueprint you may notice that it opens in a new tab. This can be configured to one's preferences, however the retard who wrote this guide suggests that you keep most of your tabs open in the same window unless you need to see multiple items at once.

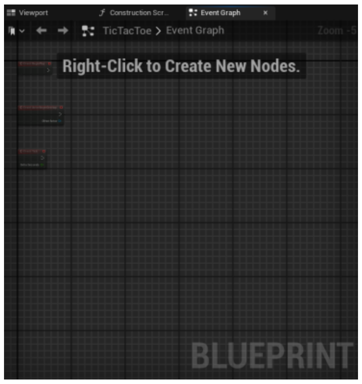
Lets take a quick look at what is on the screen now.



Pawn	
Use Controller Rotation Pitch	☐
Use Controller Rotation Yaw	☐
Use Controller Rotation Roll	☐
Can Affect Navigation Generation	☐
Auto Possess Player	Disabled
Auto Possess AI	Placed in Worl
AI Controller Class	
Input	
Actor Tick	
Start with Tick Enabled	✓
Tick Interval (secs)	0.0
Allow Tick Before Begin Play	☐
Advanced	
Camera	
Base Eye Height	64.0
Replication	
Only Relevant to Owner	☐
Always Relevant	☐
Replicate Movement	✓
Net Load on Client	✓
Net Use Owner Relevancy	☐
Replicates	✓
Net Dormancy	Awake
Net Cull Distance Squared	225000000.0
Net Update Frequency	100.0
Min Net Update Frequency	2.0
Net Priority	3.0
Physics Replication Mode	Default
Advanced	
Rendering	

1. Save Button	**2. Compile Button**	**3. Viewport**	**4. Event Graph**
5. Class Defaults -> Details	**6. Details**	**7. Functions/Macros**	**8. Variables**

1. Save Button
    a. Use it often
2. Compile Button
    a. Compile to check for errors and to make your changes apply in game
    b. If you don't compile – its not in your game
3. Viewport
    a. This is what your pawn/actor looks like. It is where we will be starting in the next section.
4. Event Graph
    a. This is where you basically make a small program which is localized on your character. You will have multiple functions here and they will do things.
        i. "BP_Third_Person_Character" is a good example of this for now if you would like to take a look.
        ii. Base Functions:
            1. Event BeginPlay – runs on start of game
            2. Event Tick – runs every game tick
                a. there is a tick for every frame, so at 60 FPS, you have 60 ticks per second

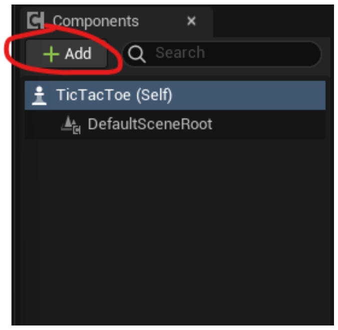
   b. 

5. Class Defaults – this is connected to #6
6. Details Pane for Class Settings and Class Defaults
   a. You won't use Class Settings very often, really only for interfaces
   b. Way too many settings to go over here, but think about it as your pawn/actor settings
7. Functions/Macros are a great way to get ugly/bulky stuff off of your event graph and keep it nice and manageable.
   a. Functions – great for items that will need to be called more than once
      i. Can be called by other actors as necessary
   b. Macros – great for items that only need to be called once or twice
      i. Can't be called externally
      ii. Every duplicated macro is essentially adding X lines of code, it is much better to reuse functions in most cases.
      iii. Great for single use cases with ugly/bulk code
8. Variables
   a. Variables
      i. Variables
   b. Can be called by external actors
   c. Ways to access externally:
      i. Cast to Blueprint, do a thing
         1. Basically impersonates another actor
         2. Costs resources as impersonated actor must always be loaded in memory
      ii. Get actor by class
         1. Great
         2. Returns an array of actors of class
         3. Cons: Have to find the actor you want
      iii. Get owner of component
         1. Assumes you have a component var to work off of
         2. Think if you hit a character in the arm hitbox and asked "Who's arm is this?"

Enough learning, back to work...

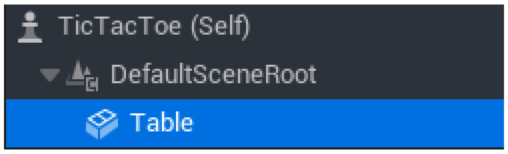
Go to viewport

Click Add Component

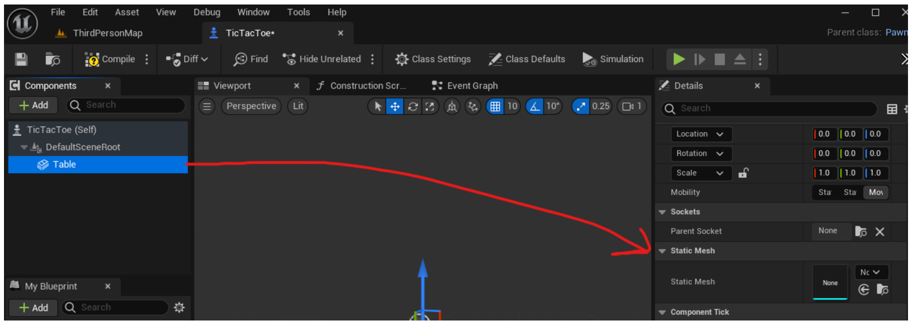


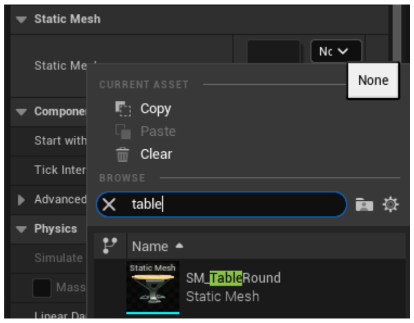
Choose “Static Mesh”

Name it “Table”



Now give it a static mesh

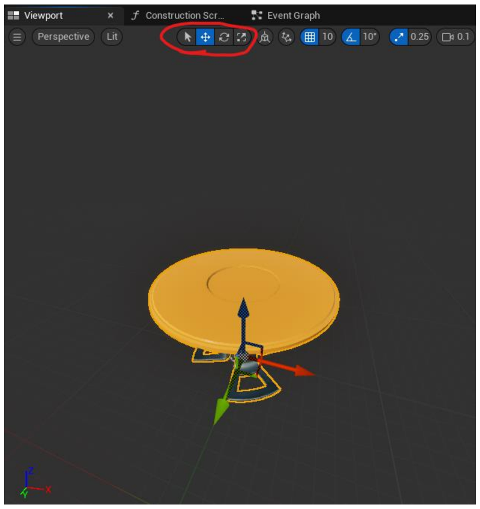




Now you have a table

Compile & Save

PDF v1.0

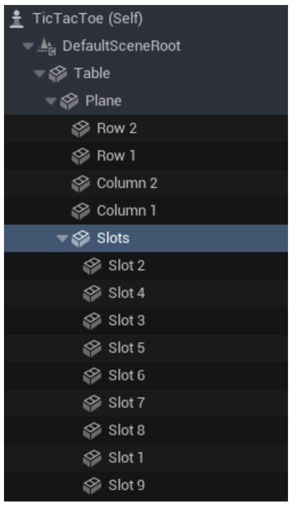


< - - Control Modes

Now for creativity, give it some more meshes and make a tic tac toe board.

You can just use Cube Meshes for everything for now, doesn't really matter.

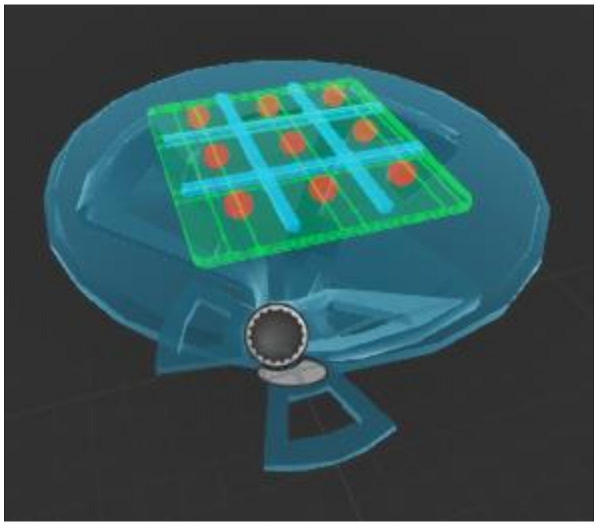
Keep in mind parent relationships like in this picture



TicTacToe (Self)
DefaultSceneRoot
Table
Plane
Row 2
Row 1
Column 2
Column 1
Slots
Slot 2
Slot 4
Slot 3
Slot 5
Slot 6
Slot 7
Slot 8
Slot 1
Slot 9

I made a plane called "Slots" with a material called "3d_Particle_Opacity_Mat" which makes it invisible. This lets me keep my slots grouped together so I can adjust the height / scale of them all at once easily.

Here is my finished product:



There are a 100 ways to skin a cat. I plan on setting the Sphere (Slot) visibility to False on "Event BeginPlay" so that I can see where the slots are in the viewport still.

I also plan on swapping the sphere mesh with a X or an O based on who chooses it. However it would be perfectly valid to set 9 slots for X and 9 slots for O and checking each time to make sure only one is ever shown for each slot. Just keep best practices in mind. In my opinion, it is always better to use modular code over easy code. At the end of the day though, who cares (me).

I am going for a Hologram Vibe for my game, so I am using translucent materials. If you want to copy mine, these are the materials I am using:

Table: DiscMaterial

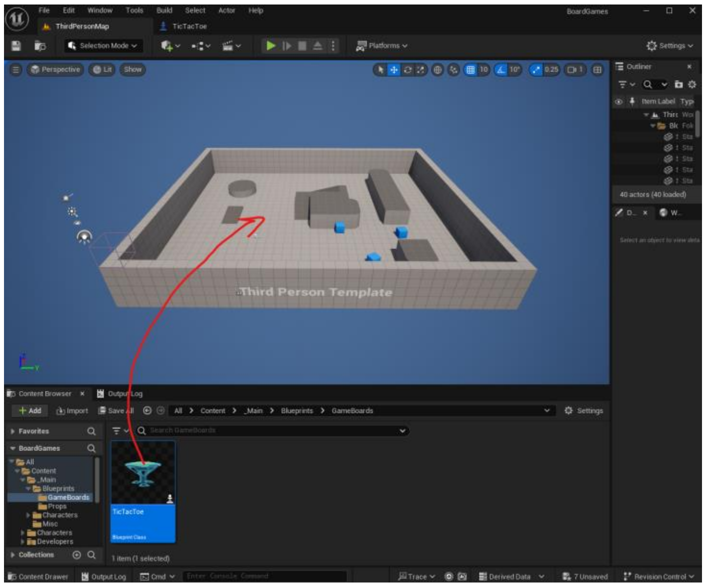
Board: MirrorPlaneMaterial

Columns: MI_FieldRadiusPreview

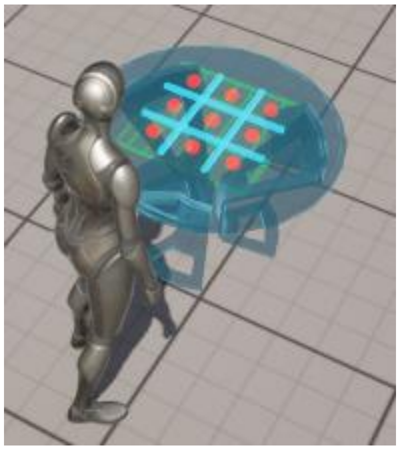
Slots: MI_SimpleUnlitTranslucent

Making Your Blueprint ~Function~

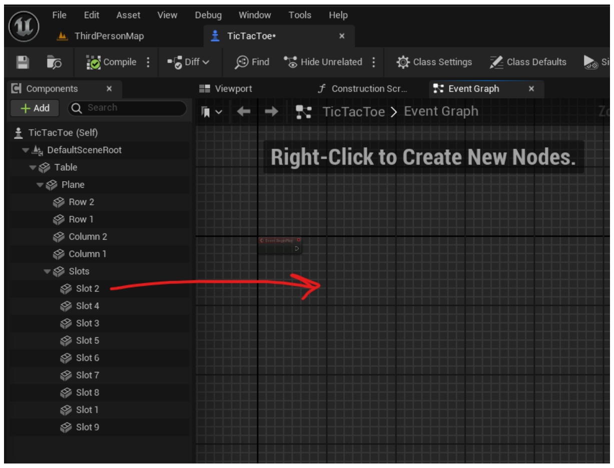
1. First of all, drag that shit into your world so you can see it



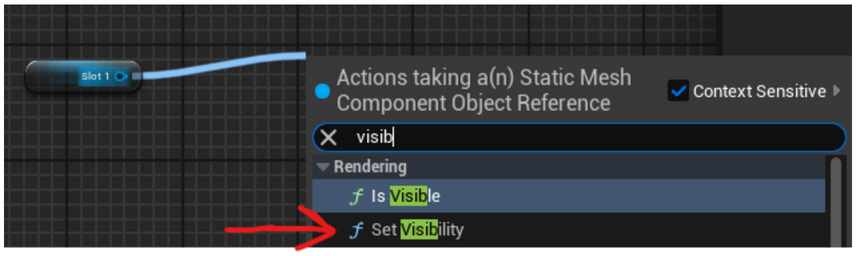
2. Click Play at the top to see it in your world



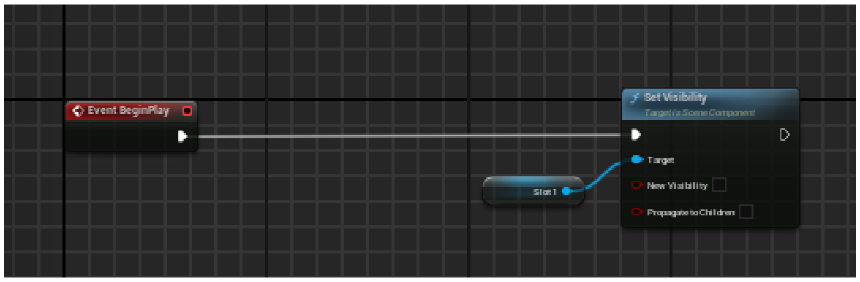
3. Okay back to what the title of this section says, lets make it do stuff
    a. Hide the slots
        i. Drag them from the components tab into your viewport



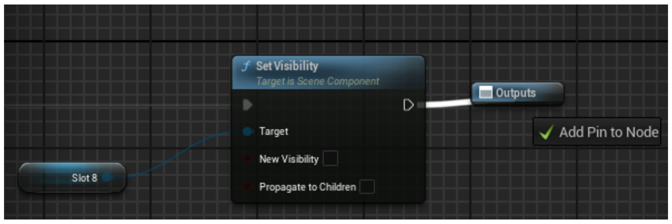
        ii. Set their visibility to False (New Visibility should not be checked)



        iii. Connect the "Execute Pins" to Event BeginPlay

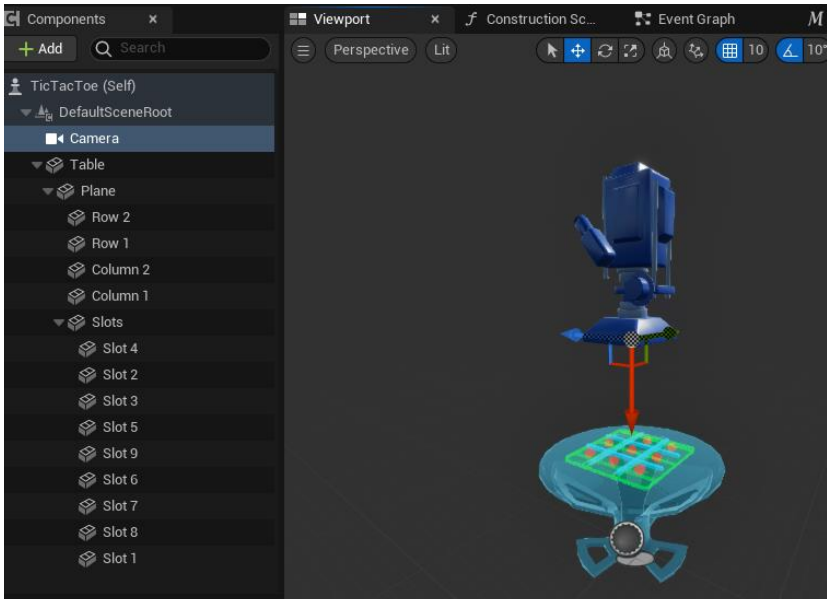


        iv. Do this for all of the slots
        v. Select them all & then right click -> "Collapse to Macro"
            1. Double click on the macro
            2. Connect the last execute pin (I forgot #9 in this pic lol)

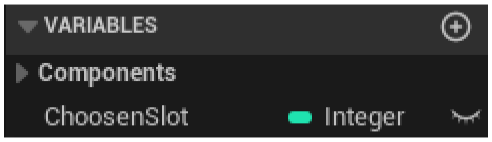


            3. Return to Event Graph and rename your Macro on the macros tab "Hide Slots"

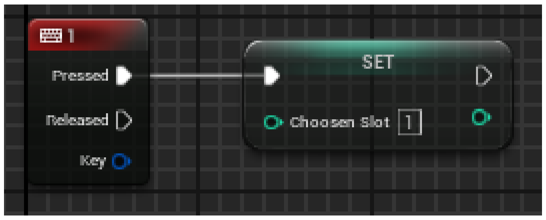
4. Hit play to see your slots be invisible in game
5. Returning to the viewport because I forgot to add this earlier... Add a camera like this:



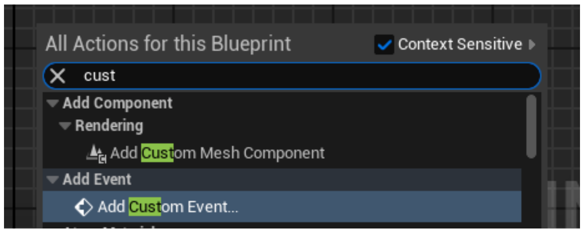
    a. You can angle it however you want
6. Back to event graph!
7. Go down a bit to some empty space
    a. Add some keyboard events, this is for the player selecting the slot they want to play
        i. Right click in empty space and type "keyboard"
        ii. Make events for 1 through 9 to correspond to each slot
    b. Add a integer variable called "Chosen Slot"



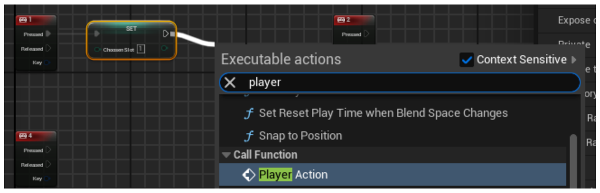
    c. For each event, set the value of chosen slot equal to the number pressed
        i. Right click in empty space and type "set chosen slot"



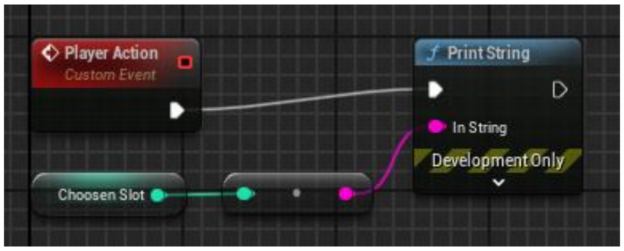
d. Make a custom event below your keyboard events and call it "Player Action"



e. Off of each set variable, run the newly created event



f. Print the current slot off of the Player Action event
   i. Get the Chosen Slot variable
   ii. Plug it into the "In String" of a print statement coming off of Player Action
   iii. The "Int to String" node will automatically appear



g. Compile & Save

Now if you hit **Play** you can see for yourself!

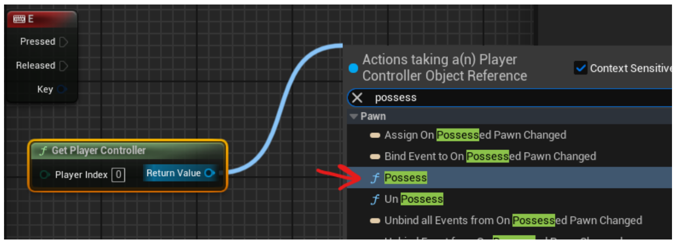
After you have done that you may be somewhat confused as to why nothing happened. Well obviously it is because ~you are not a fucking table~.

Im sorry that was mean.

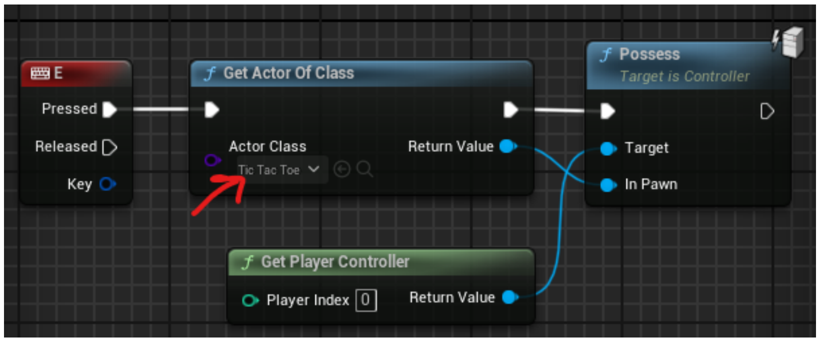
But yeah you have to become a table. Or posses one.......

Possessing Pawns

1. Navigate to your "BP_Third_Person_Character"
   a. ThirdPerson > Blueprints
2. Go down to empty space and make an event for when the keyboard input "E" is received.
   a. Just like we did earlier with the numbers
3. Get the player controller and off of the return value, call "Possess"

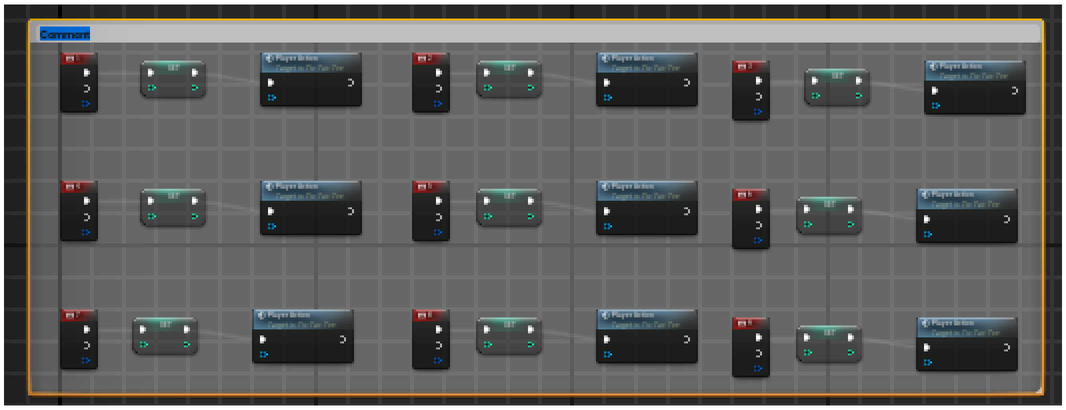

E
Pressed
Released
Key
Get Player Controller
Player Index 0
Return Value
Actions taking a(n) Player Controller Object Reference
Context Sensitive
possess
Pawn
Assign On Possessed Pawn Changed
Bind Event to On Possessed Pawn Changed
Possess
Un Possess
Unbind all Events from On Possessed Pawn Changed

4. For now we will "Get Actor Of Class" which is essentially hardcoding "E" to get the first actor of a specified class.
   a. Later we will make this more complex so that we can possess a specific table if there are multiple in the world.

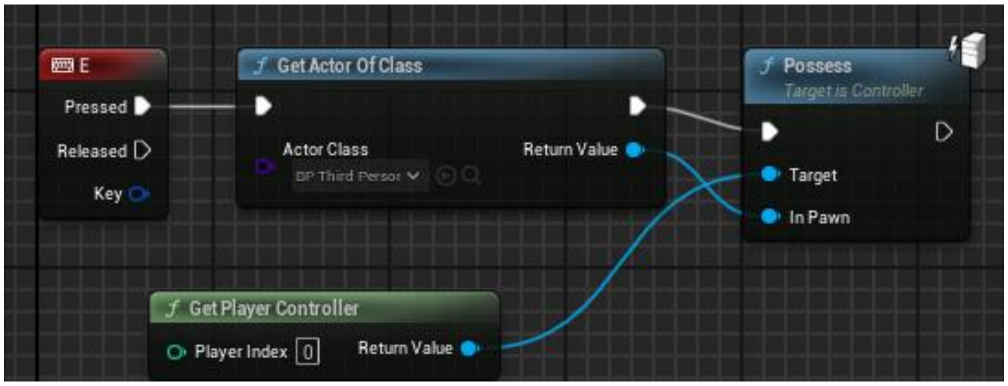


5. Compile & Save

6. Now go to your “Tic Tac Toe” blueprint again
    a. Comment your Keyboard Inputs with “C” and call it “Player Inputs”



    b. Make another keyboard input event for “E” here, this time we will possess the original player character. This will essentially let us quit the board game.



7. Compile & Save
8. Click **Play** and test it out!
    a. Now when you enter the Tic Tac Toe Pawn, you will be able to press 1-9 and it will print to the top left of your screen.

Next Steps

Planning is essential for every project, and every task. If you don’t plan, your efforts may be in vain.

So lets look at what is coming next…

“Event tick” for your Tic Tac Toe board will house a function that will wait for player input, and then figure out what to do once the player has done something.

“Player Action” for your Tic Tac Toe board will see if it is allowed to make a move, and if it is, it will make it so event tick will pick up the move and act accordingly.

This is how I am structuring my project, but there are 100 way to skin a cat :D

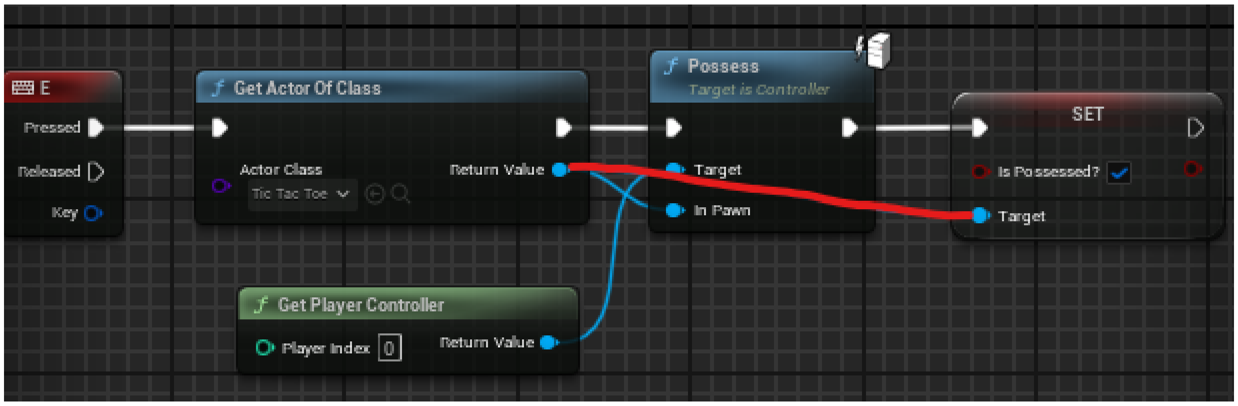
Coding – this part is long

There is no easy way to walk someone through coding. But if you are a developer, then coding should not be a foreign concept.

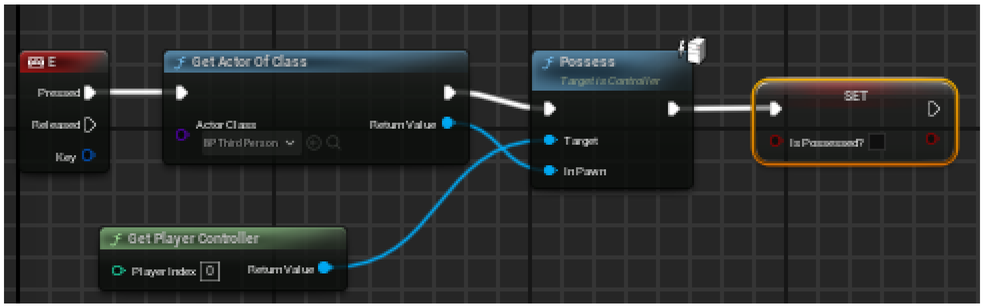
I will do my best to explain the code in a way that makes sense, following my own train of thought.

1. My first thoughts
    a. I feel like it is more important to get the turn system down between the computer and the player. This is because I will logically be checking if it is the player's turn before moving into the actual game – in the code.
2. Starting with the variables
    a. You will need to compile before setting the default values
    b. Set the default values near the bottom of the details pane when you have the variable selected.
        i. Add the following variables
            1. IsPlayerTurn?
                a. Boolean
                b. Default: False
            2. TurnMessageDisplayed?
                a. Boolean
                b. Default: False
            3. IsPossessed?
                a. Boolean
                b. Default: False
            4. IsFirstRestart?
                a. Boolean
                b. Default: True
            5. IsGameOver?
                a. Boolean
                b. Default: False
            6. HasStarted?
                a. Boolean
                b. Default: False

3. Create logic to tell the game when it has been possessed
    a. Go to your BP Third Person Character
        i. Following the red line, drag and right click.
        ii. Type "set is possessed" and it should let you set the variable
        iii. Make sure your execute pin is connected otherwise that part of the code will not run

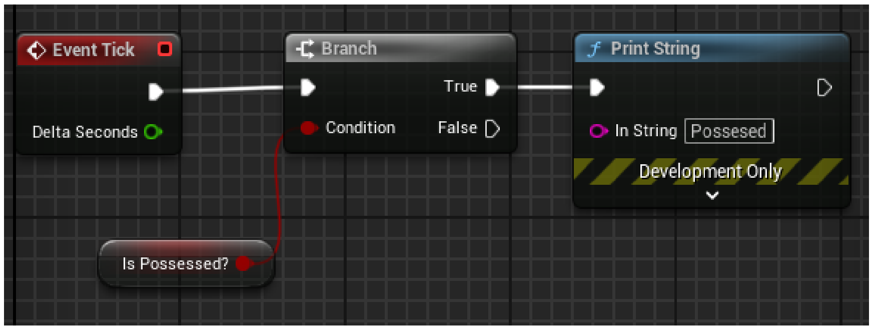


    b. Back to Tic Tac Toe
        i. Off of your "E keyboard event", set IsPossessed? To false



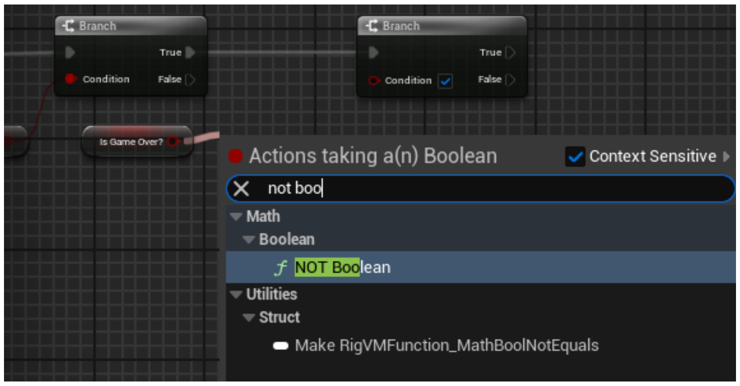
        ii. You do not have to drag off of anything this time because you are inside of the TicTacToe BP, so you are telling your own variable to be set here.
        iii. This is as opposed to the previous step where we had to reference Tic Tac Toe.

c. Now off of your Tic Tac Toe event tick, do something like this:



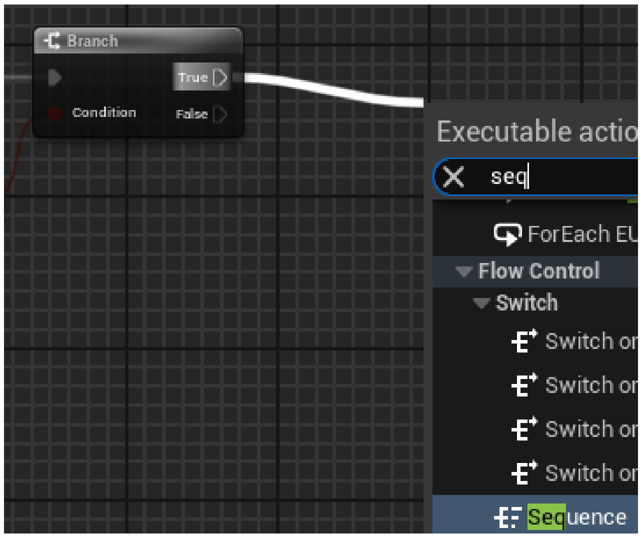
   i. Tip: You can drag off of variables in your variable pane and get/set them, this only works for variables that are owned by the BP (blueprint) you are viewing.
   ii. Now if you compile & save, you can test it out
      1. It will only print when you are possessing the Tic Tac Toe board

d. Now we are moving on to make the event tic functional
   i. The method we are going about for this game is called polling
      1. Polling means to repeatedly ask/check for input until it is received
      2. You can achieve this in multiple ways
         a. Creating a function which calls itself every 0.2+ seconds
            i. More calls will make it error as an infinite loop
         b. Going off of event tick
            i. Requires a little more logic but is probably easier for a beginner to understand because you don't have to call the event tick at the end of every branch
   ii. If the pawn is possessed, check if the game is NOT over



   iii. We are now going to define the following three game states
      1. Intro
      2. Player Turn
      3. Computer Turn

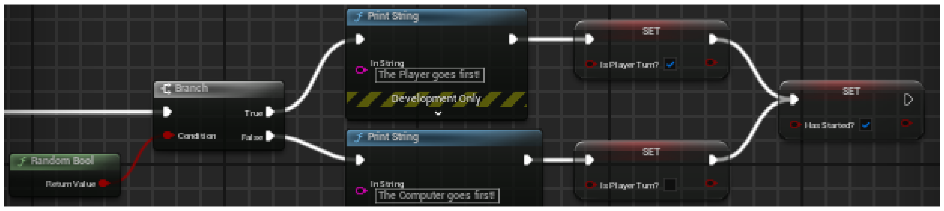
iv. Make a sequence after !(IsGameOver?) branch and come off the true with a "Sequence" node



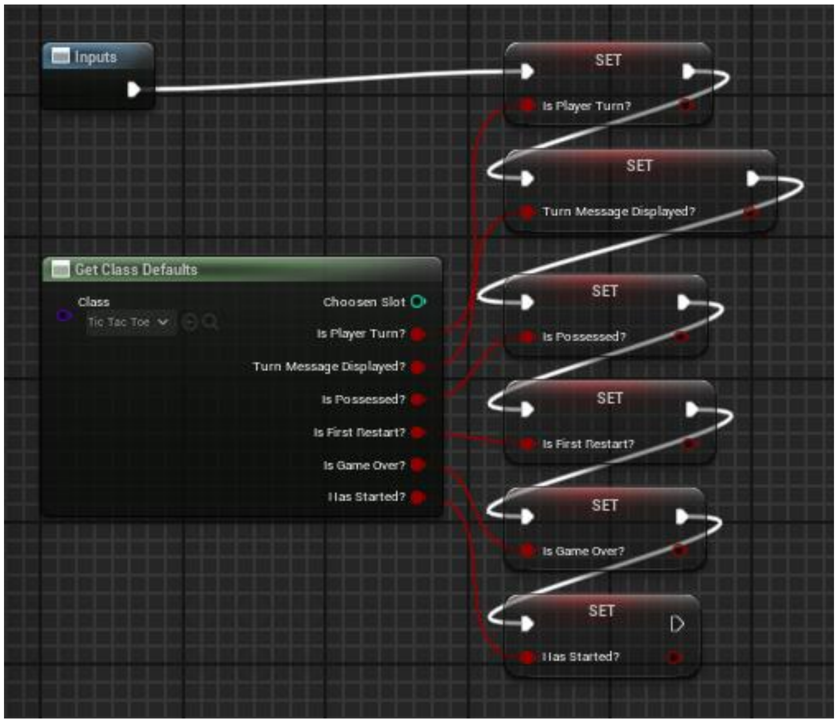
v. Add a pin so you have three output pins
    1. Keep in mind that these run in order
    2. All functions in this are single-threaded, this basically means one tiny guy in your computer is running around doing all of the work (Per loaded BP).

vi. For the first item in the sequence (intro) make a branch to check if the game Has NOT Started using the corresponding Bool variable you created
    1. Tip: You can double click on the lines (White Execute Lines for example) and get a reroute node. You can use this to make your code prettier.

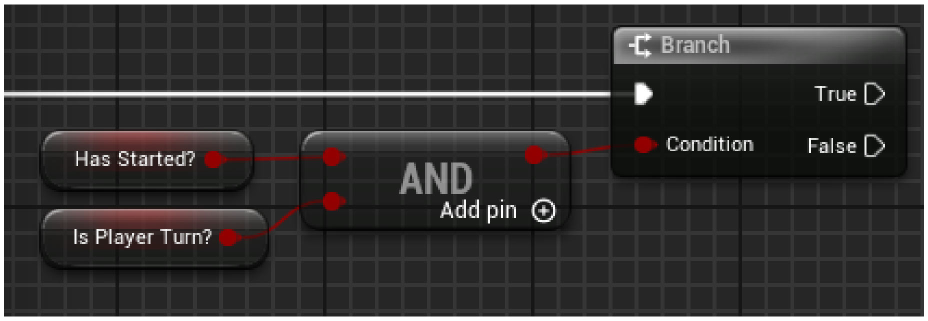
vii. If the previous branch is TRUE, then run a Print String node saying "Welcome to Tic Tac Toe"
    1. After this get a "Random Bool" node with a branch
    2. Off of the branch, decide if it is the player's turn or not
    3. Make both branches set HasStarted? to TRUE



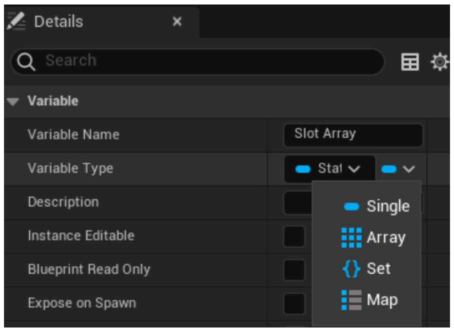
e. Back down to “E event”
    i. Collapse the IsPossessed? set node you just created into a macro by right clicking
        1. Name it “Wipe Board”
        2. Get the node called “Get Class Defaults”
        3. Set all variables to their default values
        4. This means that every time you re-enter the board, it will be reset :D
        5. You will have to come here later and reset more variables once we have created them



f. Back to Event Graph -> Event Tick -> Sequence
    i. Off of Sequence “Then 1” we will create the logic for the player’s turn
        1. Create a branch with an “AND Bool” node
            a. Check if the game Has Started
            b. Check if it is the Player’s Turn



g. Make another branch to check if the Turn Message has been Displayed
    i. When should you use a NOT Bool?
        1. I like to use NOT Bools whenever I want to do something first that comes off of the False Branch, or when the branch's purpose is to do the opposite of a variable.
        2. In this case I want to print a message if the player has not been notified that it is their turn. So I am using a NOT Bool here
h. Coming off "TurnMessageDisplayed?" -> False
    i. Make a print statement that says "Press 1-9 on your keyboard to choose your next square"
    ii. Set "TurnMessageDisplayed?" to TRUE
i. Moving on to the next item in the sequence "Then 2", we will set up the computer turn
    i. Just like the previous step, we will make an AND bool branch that is exactly the same except for the IsPlayerTurn? Will need to be modified by a NOT bool
    ii. I want to have a hard mode later, so I am going to....
        1. Create a new variable called "IsEasyMode?" that is a bool with a default of TRUE
        2. Create a branch for this variable coming off of the true of the previous branch
    iii. Add a variable to hold the slots and their owner
        1. Slot Map
            a. Type: Static Mesh Component Object Reference
            b. With the variable selected, go to the details pane on the right and change the variable from a single to an Map.



Details

Variable

Variable Name: Slot Array

Variable Type: Stat

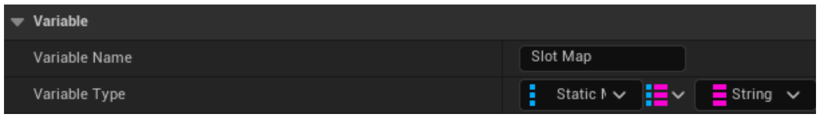
Description

Instance Editable

Blueprint Read Only

Expose on Spawn

                i. Maps are made of Key-Value Pairs. They are like arrays, but the Key part should not be changed, it is like having a second index value that you can search by.
            c. Make the Value be a string



Variable

Variable Name: Slot Map

Variable Type: Static M | String

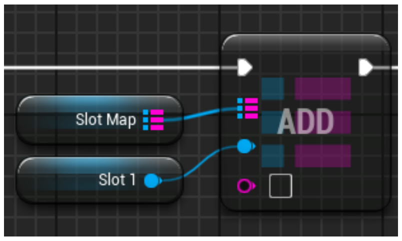
iv. Build the slot map

1. On “Event BeginPlay” -> Hide Slots Macro

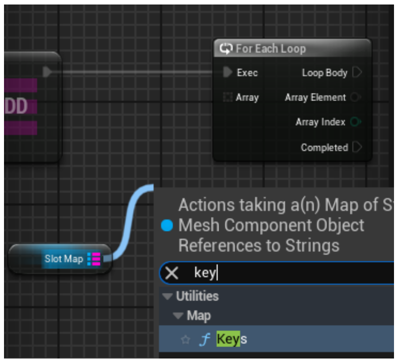
a. We are going to rewrite this so it builds our map and sets visibility more efficiently

b. Delete everything except for the Inputs/Outputs

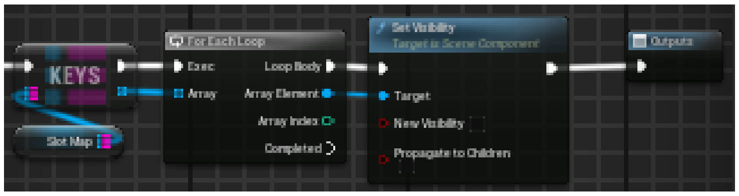
c. Do an “Add Map” for each slot mesh



d. At the end, do a For Each loop with the slot map keys

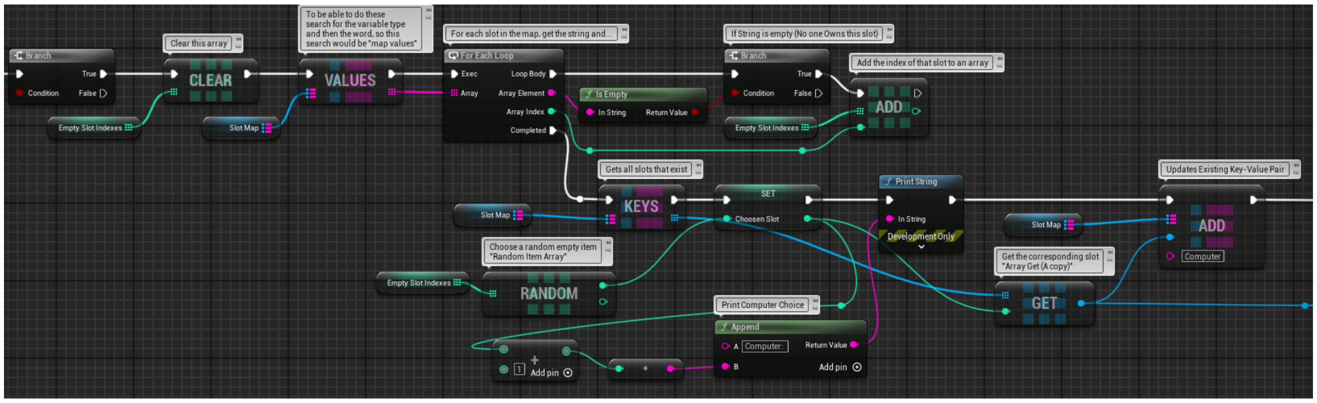


e. Set this visibility for each slot



f. Compile & Save

v. Have the computer choose a spot
   1. Get reduce the map to keys that do not have associated string values
      a. Aka find empty spots
      b. Make a new variable called “EmptySlotIndexes” which will be an Integer Array
      c. Add the following logic, there are comment bubbles to explain what is happening in each node


Clear this array

To be able to do these search for the variable type and then the word, so this search would be "map values"

For each slot in the map, get the string and...

If String is empty (No one Owns this slot)

Add the index of that slot to an array

Gets all slots that exist

Choose a random empty item "Random Item Array"

Print Computer Choice

Get the corresponding slot "Array Get (A copy)"

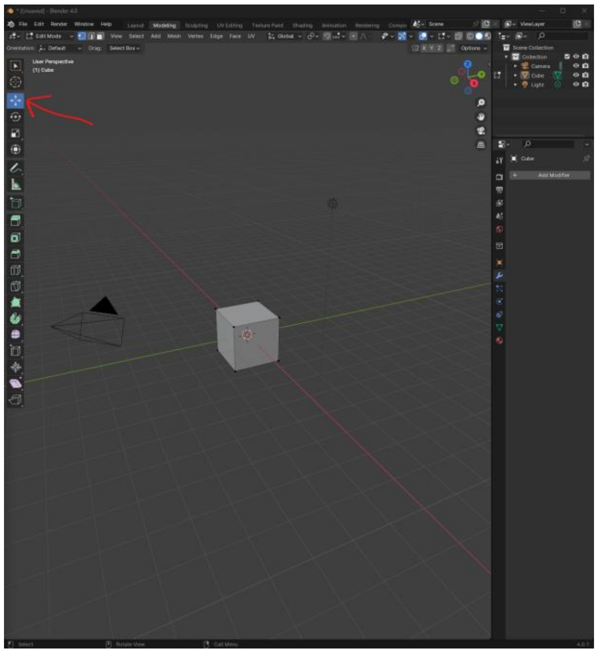
Updates Existing Key-Value Pair

vi. Compile & Save

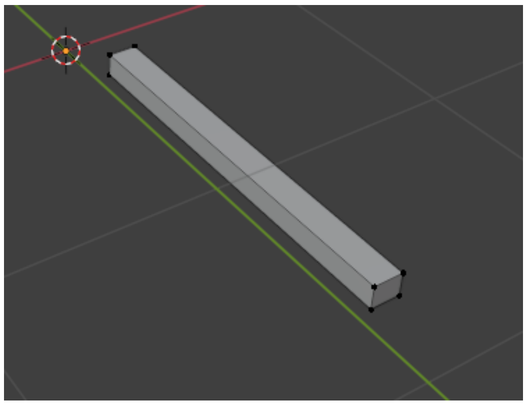
Game Piece Meshes

We will now create studio-quality X and O meshes for the board because I am tired of explaining code. :D

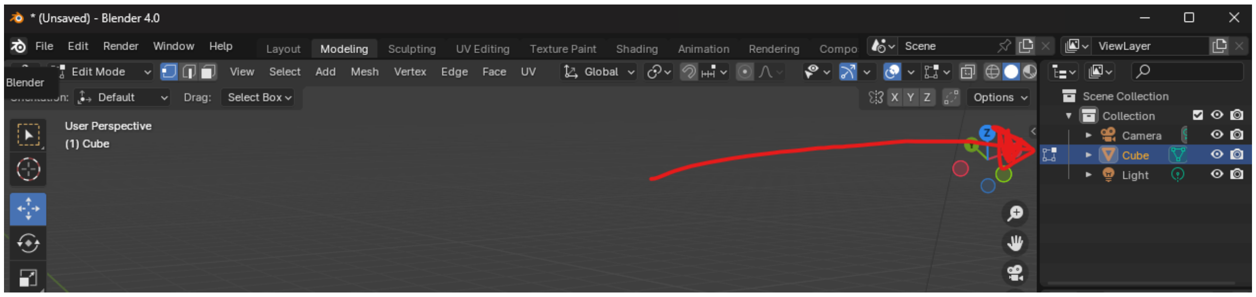
1. Open Blender!



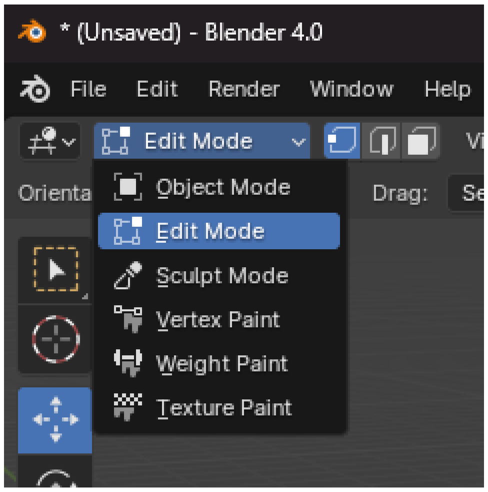
2. Now click on a vertex (small black dots)
3. Now move it, your goal is to make a rectangular prism
    a. Rotate camera with pushing down middle mouse and drag
    b. You can select multiple vertexes at once
4. Once you have your stick…



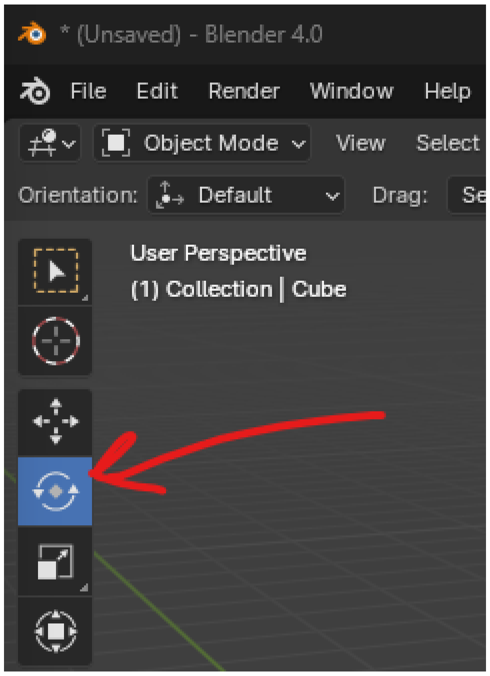
5. Make another Stick!



6. Right click, copy, right click, paste
7. Change to object mode

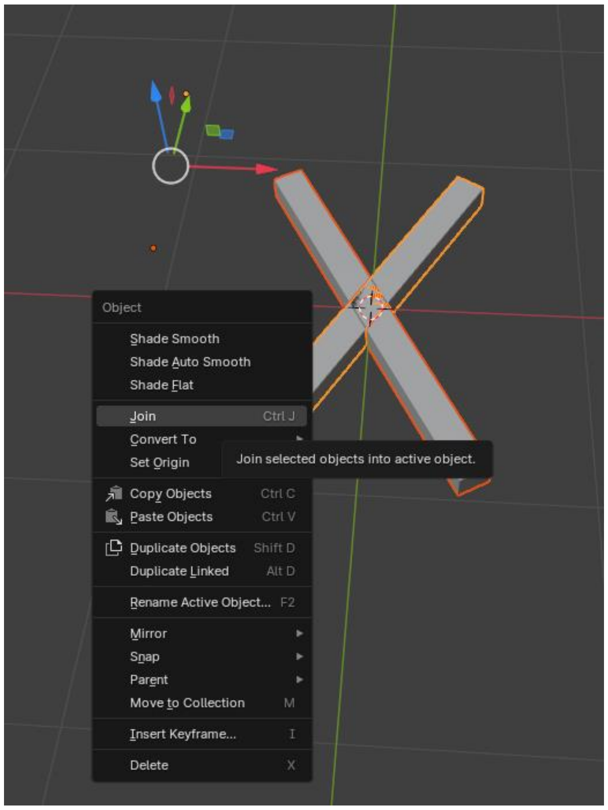


8. Rotate/Move it and make an X



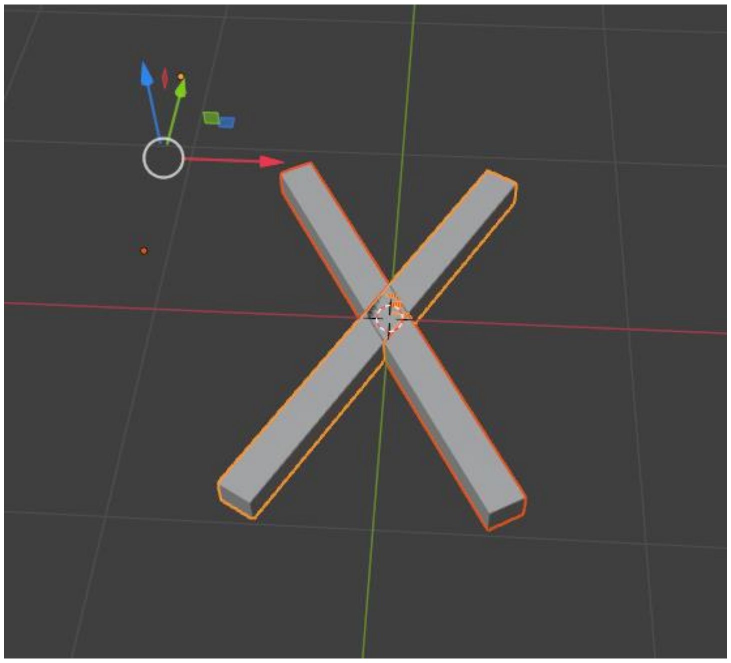
9. Select them both in the top right and then move/rotate them back to the about 0,0,0 (the center)

10. Once they are both in the correct place, select them both and click “Join”



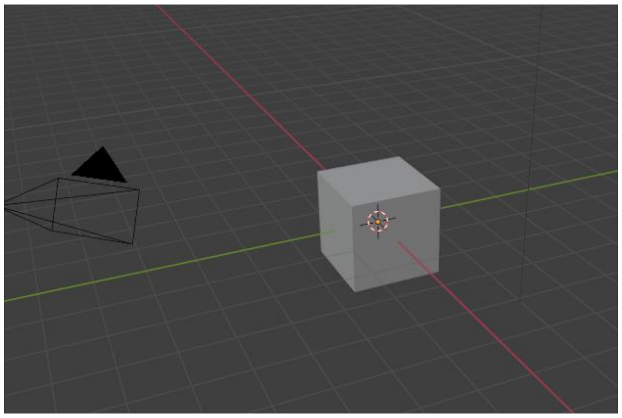
Join selected objects into active object.

11. Admire your work

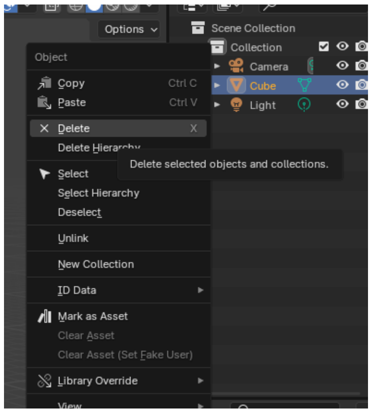


12. Export it to your downloads folder
    a. File > Export > FBX
    b. Name it “x-shape.fbx”

1. Now lets make an O
2. File > New > General

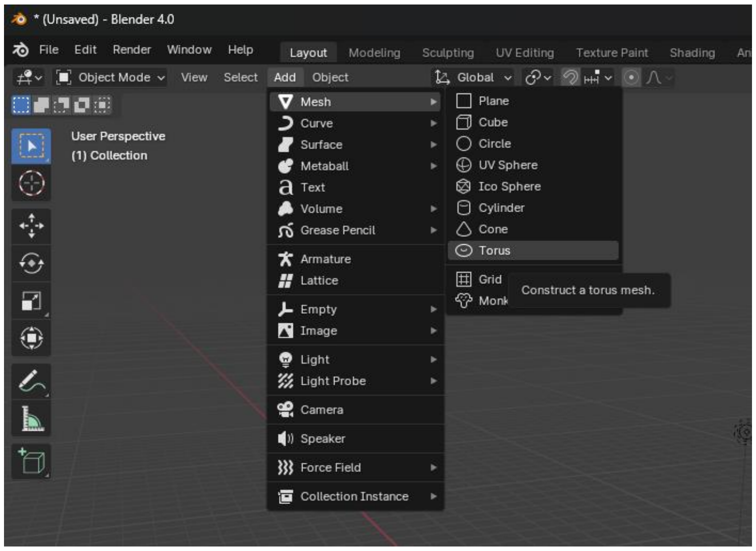


3. Delete the cube



Delete selected objects and collections.

4. Add a Torus Mesh



Construct a torus mesh.

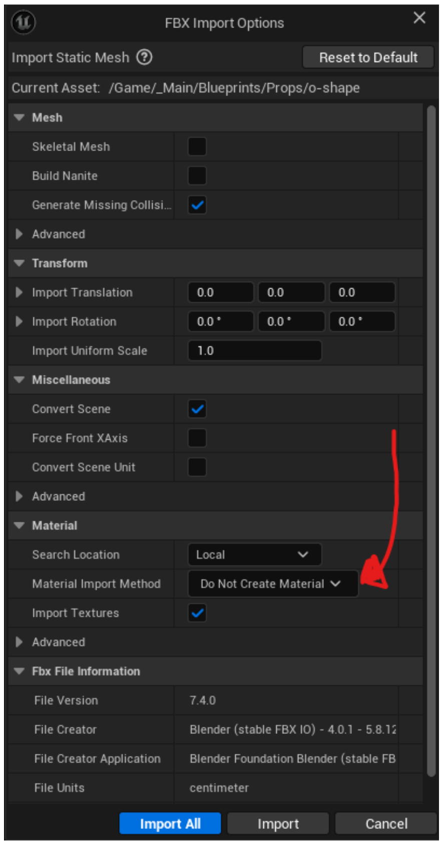
5. Export just as before named “o-shape.fbx”

Importing Your Meshes

Back to Unreal Engine! ~~I don’t like blender :(~~

~~Its hard and I don’t know how to use it well~~ Im a pro at blender by the way, im sure you can tell

1. Navigate to “_Main>Blueprints>Props”
2. Click Import in the Top left of the content browser
3. Import Both Meshes that you created
    a. Select “Do Not Create Material” for the import options



FBX Import Options

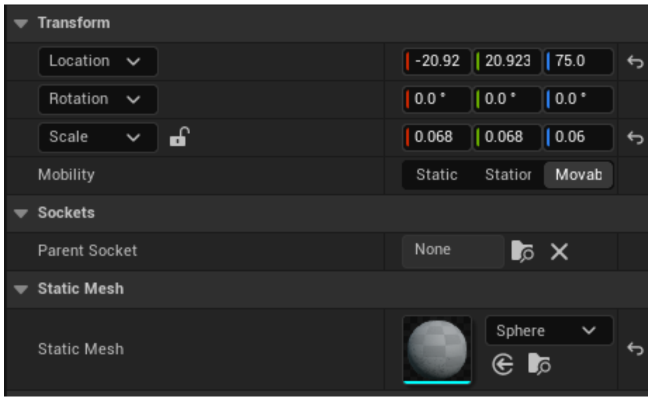
Import Static Mesh

Current Asset: /Game/_Main/Blueprints/Props/o-shape

Mesh	
Skeletal Mesh	☐
Build Nanite	☐
Generate Missing Collisi...	✓
Advanced	
Transform	
Import Translation	0.0 0.0 0.0
Import Rotation	0.0 ° 0.0 ° 0.0 °
Import Uniform Scale	1.0
Miscellaneous	
Convert Scene	✓
Force Front XAxis	☐
Convert Scene Unit	☐
Advanced	
Material	
Search Location	Local
Material Import Method	Do Not Create Material
Import Textures	✓
Advanced	
Fbx File Information	
File Version	7.4.0
File Creator	Blender (stable FBX IO) - 4.0.1 - 5.8.12
File Creator Application	Blender Foundation Blender (stable FB
File Units	centimeter

4. Give the newly created game pieces the material texture you want and then save

5. Go to your TicTacToe BP, we now need to check the sizing for our newly created objects
    a. For reference, my slot spheres are modified by these scale settings
        i. This means that the resizing later will multiply based on the value here and the one we set further down



Transform

Location	-20.92	20.923	75.0	
Rotation	0.0 °	0.0 °	0.0 °	
Scale	0.068	0.068	0.06	
Mobility	Static	Statior	Movab	

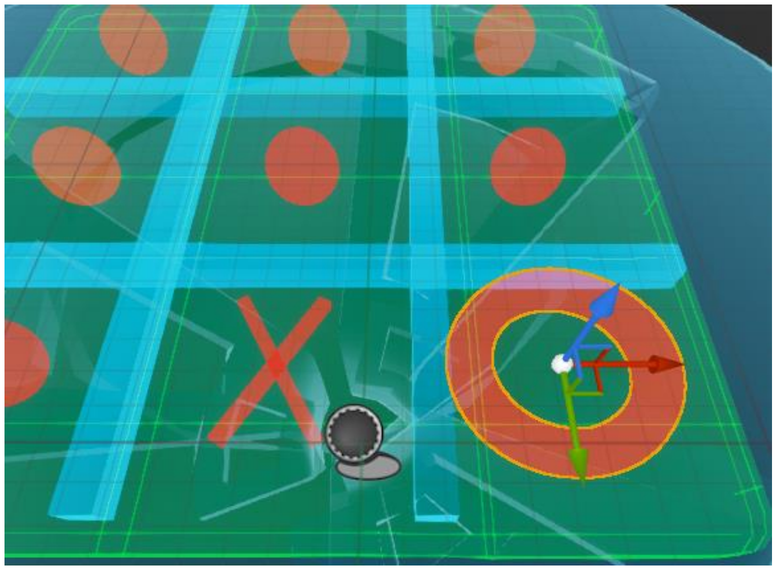
Sockets

Parent Socket	None

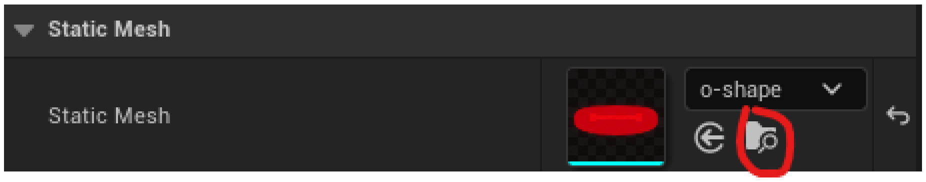
Static Mesh

Static Mesh	Sphere

    b. Replace one of your spheres with the each shape to check the sizing

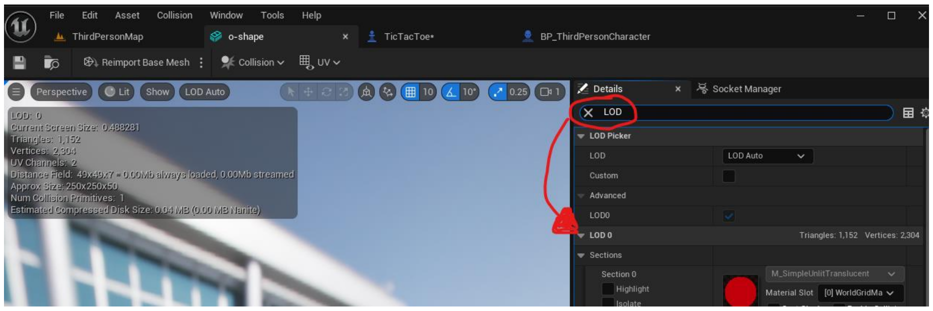


c. My X is fine but my O is a bit too large, instead of changing the size here, go back to the prop mesh
    i. This is a shortcut to get you there faster



Static Mesh	o-shape

    ii. Once you open your o-shape, search for LOD, we are looking for LOD 0



```
LOD: 0
Current Screen Size: 0.488281
Triangles: 1,152
Vertices: 2,304
UV Channels: 2
Distance Field: 49x49x7 = 0.00Mb always loaded, 0.00Mb streamed
Approx Size: 250x250x50
Num Collision Primitives: 1
Estimated Compressed Disk Size: 0.04 MB (0.00 MB Nanite)
```

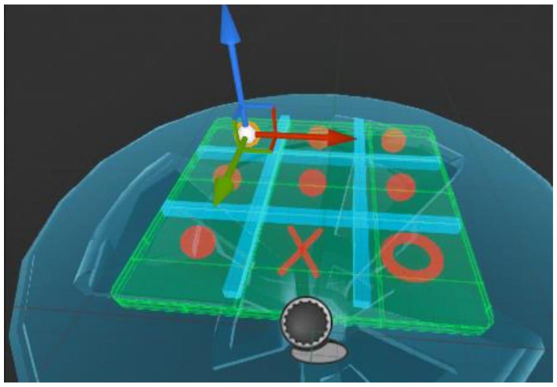
LOD

LOD Picker	
LOD	LOD Auto
Custom	☐
Advanced	
LOD0	✓
LOD 0	Triangles: 1,152 Vertices: 2,304
Sections	
Section 0	M_SimpleUnlitTranslucent
Highlight	Material Slot [0] WorldGridMa
Isolate	

    iii. In LOD 0, you can mess with the build scale of your object
        1. I found that these values worked well for me

Build Scale	0.7	0.7	0.6

    iv. Once you are done, hit "Apply Changes" and check the TicTacToe BP as it should have updated instantly.
    v. Make changes as necessary

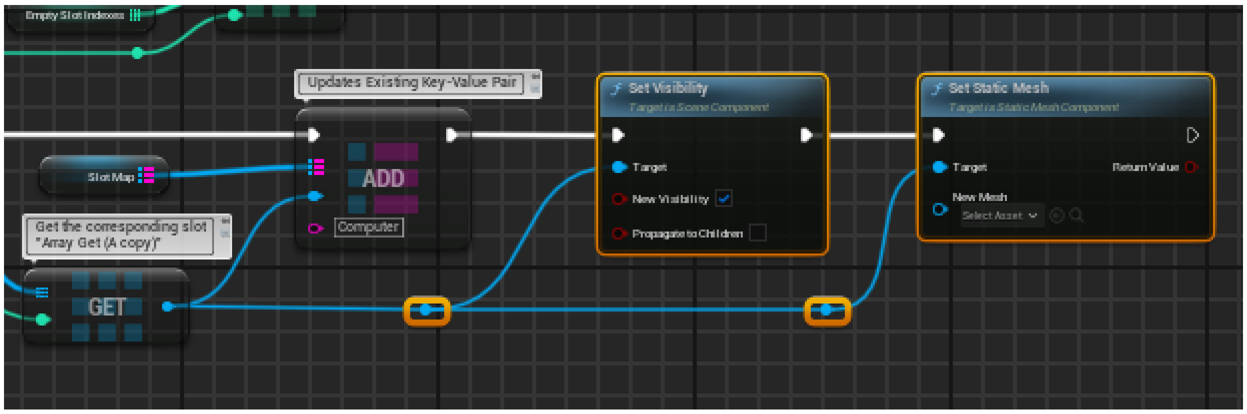


d. Change these meshes back to spheres, or whatever. Because you will change them later in the code.

Making The Board Work

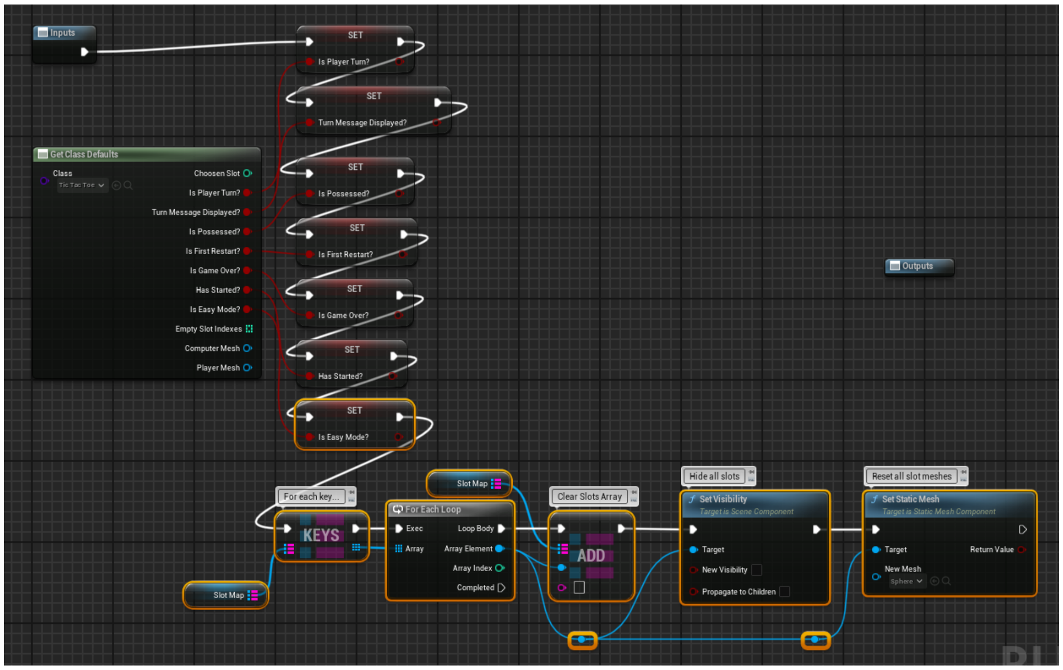
Now we return to working on the computer's turn

1. After we choose where the computer wants to play, we need to set the slot mesh to the computer's X or O and make it visible.
    a. Tip: you can use one output node for multiple inputs in many cases
    b. Tip: you can use reroute nodes to make your code look nicer :D

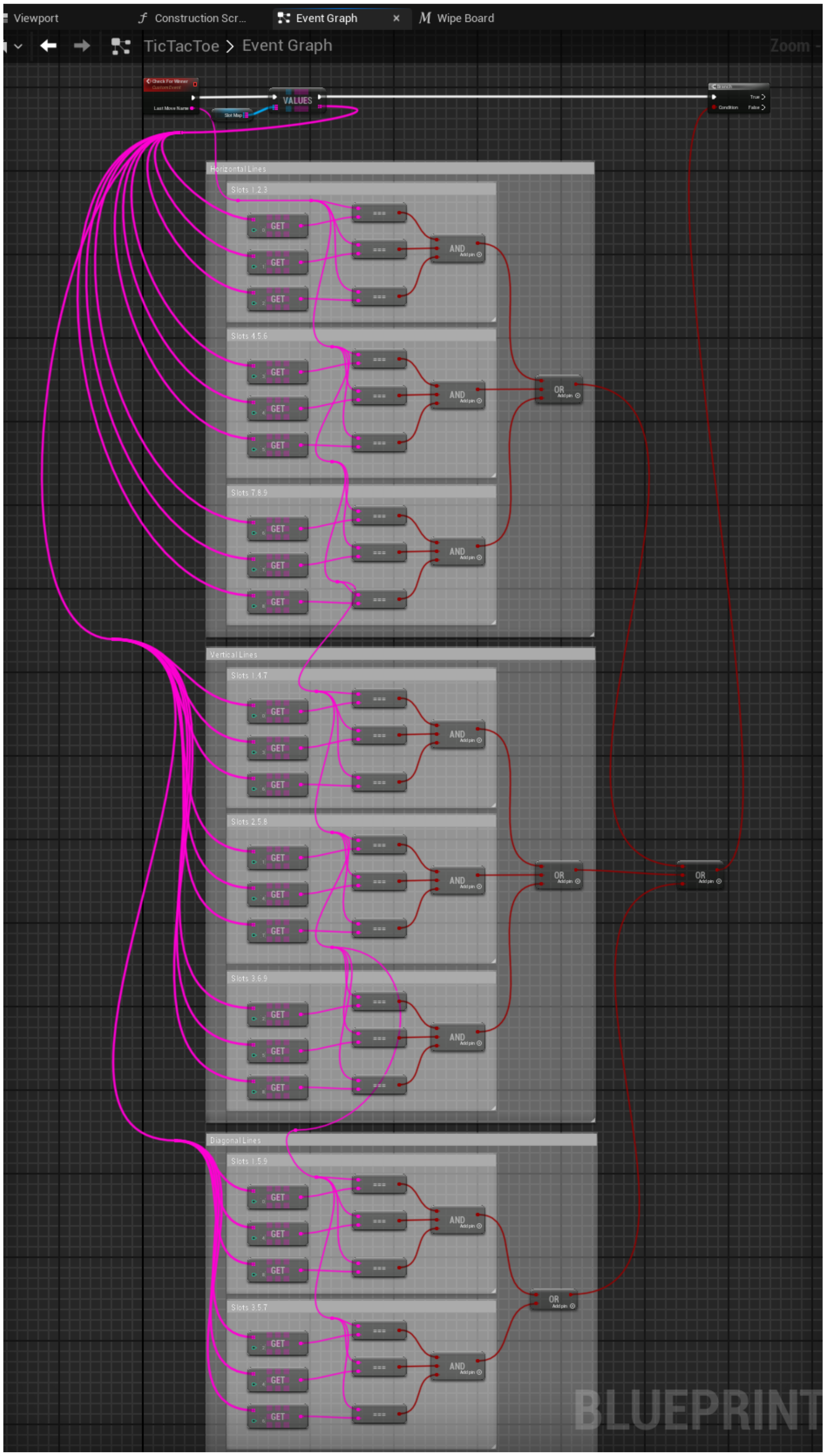


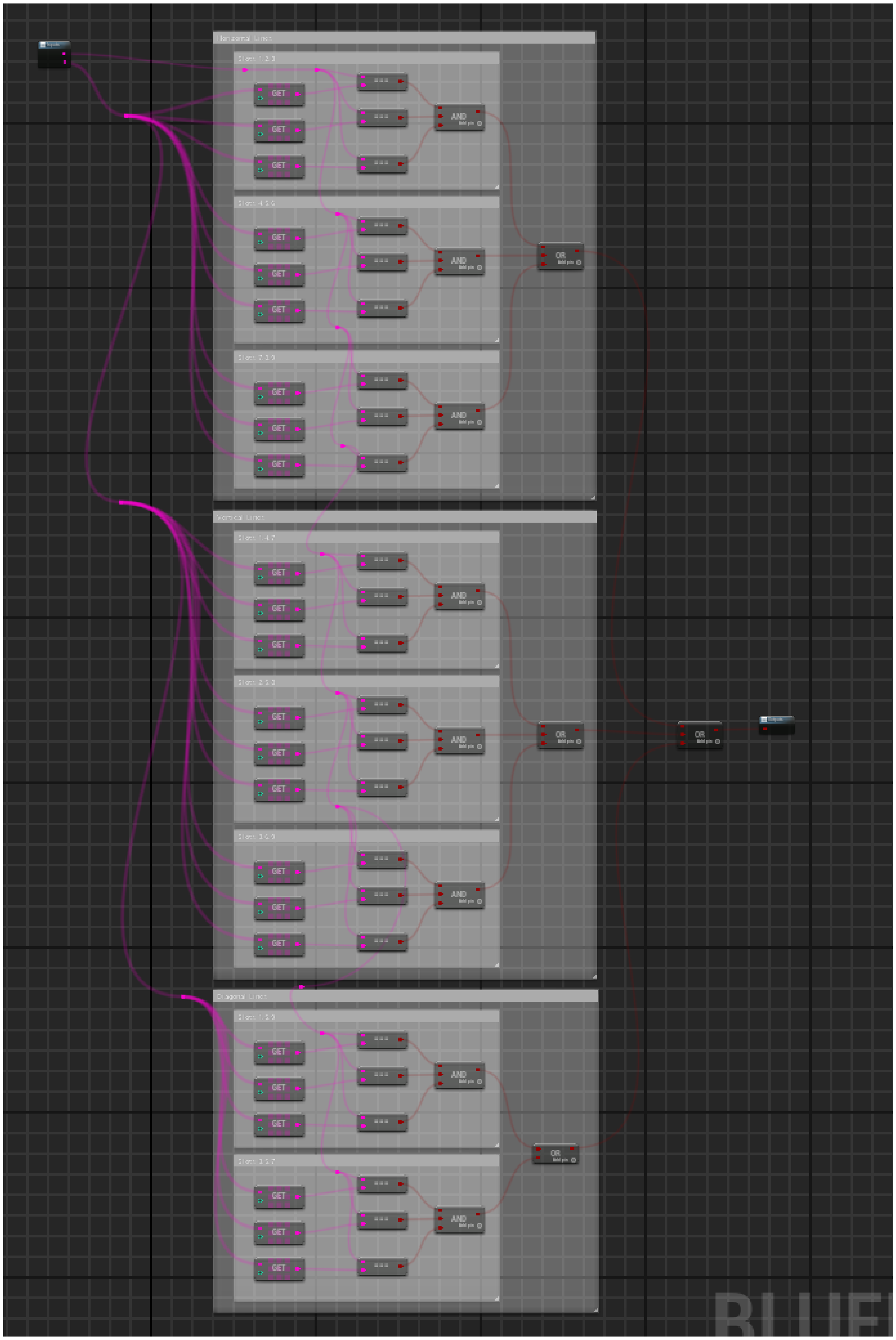
2. Create New Variables
    a. In the "Set Static Mesh" node...
        i. Right click on "New Mesh" and click "Promote to Variable"
        ii. Name it "Computer Mesh"
        iii. Choose the o-shape mesh you created earlier
    b. Create a new variable
        i. Name it "Player Mesh"
        ii. Choose the x-shape mesh you created earlier
        iii. Tip: Instead of searching for the type of variable, you can disconnect the Computer Mesh and promote the "New Mesh" to a variable again. Just make sure to reconnect computer mesh after!

3. Navigate to your “Wipe Board Macro”
    a. Make the highlighted changes

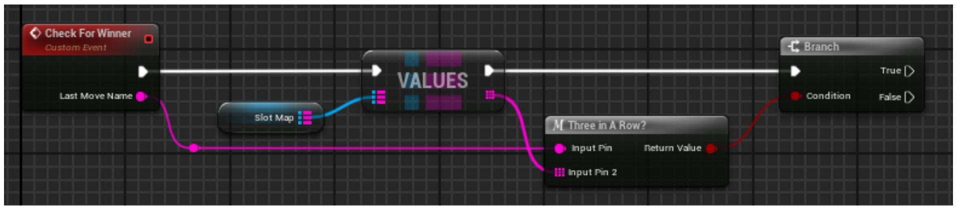


    b. This will make it so each time you press “E” to exit the game, your game should reset
4. Back to the computer turn logic....
    a. After we set the static mesh for the slot, we want to check if the computer won
        i. Go down and find some empty space, create a custom event called “Check For Winner”
        ii. Add an input String to the function from the details pane
            1. Name it “Last Move Name”
        iii. Now we have to hardcode the relationships for the array map... Lame
            1. The next page has an image showing what this looks like
            2. If you can make it look better, go crazy.
                a. I was tempted to make it a macro that takes in the string array, and the ‘Last Move Name’ and outputs a Boolean TRUE/FALSE but I already finished it by the time I thought of that. EDIT: I did make it a macro, check 2 pages down for pic
                b. If you do this, with your macro open, you can just drag pins onto the input/output and it will add the vars for you.
                c. Or you can just go to the details pane and add the vars manually to the input/output.
                d. Keep in mind that these input/output vars are not linked to your BP variables, if you want to use something, add it as an input, if you want to return something, add it as an output.


Viewport
Construction Scr...
Event Graph
Wipe Board
TicTacToe > Event Graph
Zoom
Check For Winner
Last Move Name
VALUES
Slot Map
Branch
True
False
Condition
Horizontal Lines
Slots 1,2,3
Slots 4,5,6
Slots 7,8,9
Vertical Lines
Slots 1,4,7
Slots 2,5,8
Slots 3,6,9
Diagonal Lines
Slots 1,5,9
Slots 3,5,7
GET
===
AND
OR
Add pin
BLUEPRINT


Horizontal Lines
Slots 1 2 3
Slots 4 5 6
Slots 7 8 9
Vertical Lines
Slots 1 4 7
Slots 2 5 8
Slots 3 6 9
Diagonal Lines
Slots 1 5 9
Slots 3 5 7
GET
AND
OR
BLUEF

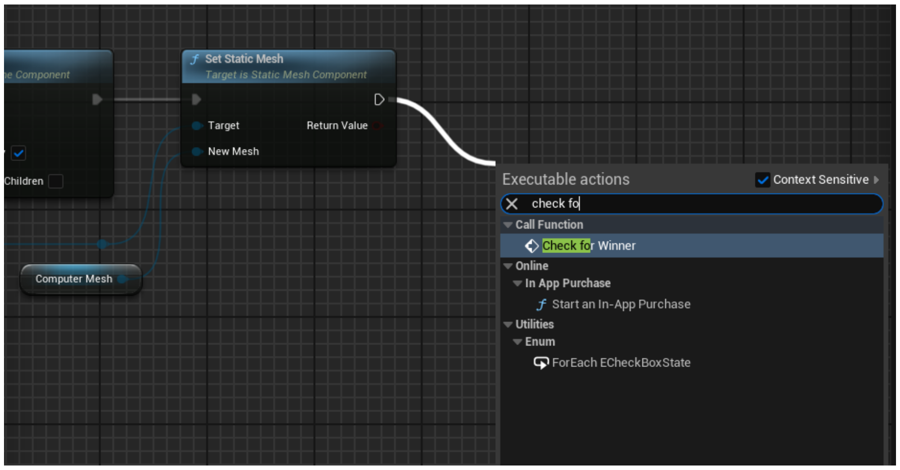
b. After making it a macro, it looks like this from the event graph:



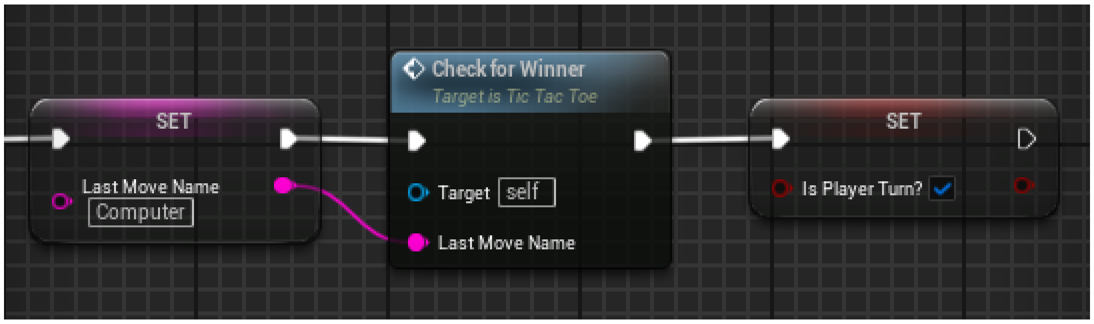
c. Now from the branch, if there is three in a row, then set IsGameOver? to TRUE

d. Next, going back to the computer logic, (if no winner) we want to tell the game that it is the player's turn

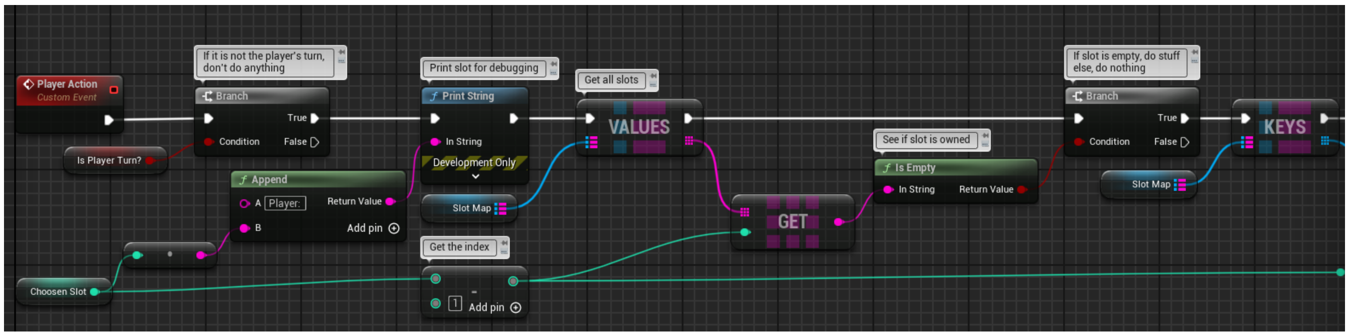
i. First check for winner after the computer makes its move

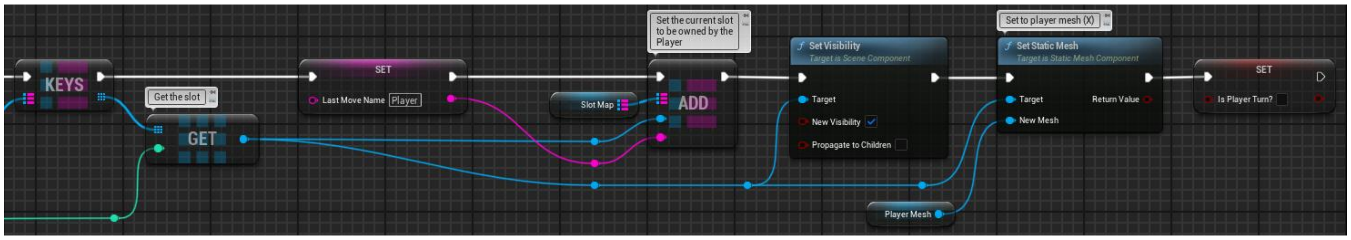


ii. Promote "Last Move Name" to a variable and set IsPlayerTurn? to TRUE

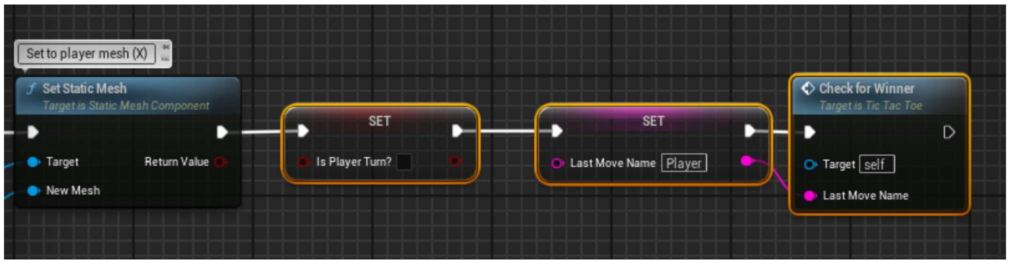


Player Turn



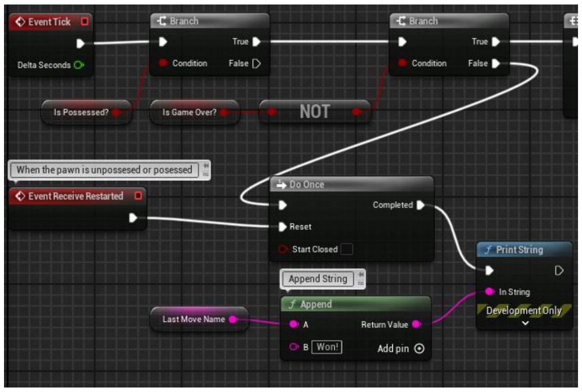


1. Update your “Player Action” event as above.
2. After the “Set Static Mesh” node:



3. On your Player Action Event, go to the branch that sees if the slot is unowned, off of the FALSE branch, print “Slot Already Filled!”

4. Go up to your event tick where you have the branch for !(IsGameOver?)
5. Under the false condition, print the winner from your “Last Move Name” variable
    a. Keep in mind that Event Tick runs every tick so if you don’t set up the do once and event receive restarted like I did, it would just spit out “_ Won!” over and over for infinity



    b. After printing the winner, set “IsPlayerTurn?” to FALSE

6. Compile & Save
7. Go test it out!