Item Inspection

**Look At**

Open the third person character / first person character BP and in the viewport add a scene component and call it ItemGrip. Attached it to the Camera so that it moves in the direction of the camera. I’m using a scene component for now but we can also use the arm mesh or any other component. It’s basically how we want the object to be positioned when we look at it. Adjust the ItemGrip so that it’s in the right focus of the camera.

Next, we go to the folder and create a blueprint interface and name it Interface\_Interface. An interface is a set of functions that can be applied to classes. The first function we make is called Look At and the input for it is the character that looks at the interactive object. In our case it’s of type the third person/ first person/ just player character and let’s name the input as Player Character.

Add another function called Interact With. It also takes an input – the character that interacts with the object. The name of the input is Player Character and it’s of type first person/ third person character.

The way this works is we have an item parent and many child items. We will be picking up the child items while storing the bulk of the code in the item parent. Create a new folder called items and inside it make a BP of actor type and name it ItemParent. To this actor we need to assign the Interact\_Interface. Click the class settings and in the interfaces section on the right hand side click the add button and choose the Interact\_Interface. Go to the event graph and add Look At and Interact With event nodes.

Next, we need to find if the player is looking at something and if yes, then what it is. Open the third/first person character BP and make a function called Check Look At and extend it and add a line trace by channel node. Set the trace channel as camera channel and the start point is to come from the player’s camera so drag in a reference to the player’s camera and extend it and add a get world location node and hook the value to the start pin. From the camera reference add a get world rotation node and extend it and add a get forward vector node and hook the return value and add a vector \* int node and set the other pin as 250 and extend the return value and add a vector + vector node and to the first pin hook the world location node’s output pin and then hook the resultant pin to the end point. Basically, for the endpoint we take the direction in which the camera is pointing at and multiply a value so as to make the vector cover some distance in the direction of the starting point.

Split the out hit pin and extend the hit actor and add a does implement interface and choose the Interact \_Interface. Extend the return value and add a branch node and hook its input exec pin with the line trace by channel node’s output exec pin. Make a new variable called LookAtActor of type actor. Extend the false pin and add a set LookAtActor. The input pin will be unconnected so as to set the variable back to null. Before we set the value to the variable for the true case, we need to first check if the player is looking at a different object than the one, he was looking at previously. So, we get a reference to the LookAtActor and add an == node and to the other pin hook the hit actor. Extend the output pin and add a branch node and connect its input exec pin with the true pin. Then extend its false pin and hook it to the set LookAtActor node and connect the hit actor pin to the LookAtActor pin. Extend the output pin and add the Look At fn node and extend the Player Character pin and add a self node. Add an output to the Check Look At fn called LookAtActor of Actor type and extend the Look At fn node’s exec pin and add the return node and to the LookAtActor pin add a reference to the LookAtActor variable. Extend the false branch’s set LookAtActor node and add a return node to that too. Hook the set LookAtActor output pin to the return node’s input pin.

The idea is to obtain a ray trace and if it collides with an object we check if it’s an interactable object then we check if it’s a new object then we store the object in the variable.

Go back to the event graph and add an event tick node and extend it and add the Check Look At fn node. So now with every tick we check if the player is looking at an object. For debugging go back to the line trace node and enable line trace for debugging. So now if you run you’ll see a red mark appearing periodically on the screen.

**Pick Up**

The goal here is to make the looked object come to the player’s hands or the ItemGrip in this case. The Check Look At fn in the character BP returns the object that the player looks at. On pressing the appropriate key the object is transported to the ItemGrip position.

Open project settings and add an input key binding named Interact. Commonly we assign the E key for keyboards and Gamepad Face Button Left for interacting with objects so enter that as the input key. Add another button for dropping items as well so make a new input action called Cancel and add the R key for keyboard and gamepad face button right if you want. If you want you can use the same key for both Picking and Dropping and I think that’s better.

The algorithm is to get the object the player is looking at when E is pressed. If the fn returns a valid output then we check if the object is already with the player if yes then don’t pick it interact with it if no then pick it up.

Go to player character and add the Interact node. Add a reference to the Look At Actor fn and right click on it and click Convert to Validated Get. This is to make sure the fn returns a valid output. Extend the pressed and hook it to the get node’s input exec pin. Extend the Look At Actor pin and add an Interact With node (Target is Interface\_Interface) and hook its input exec pin to the get node’s is Valid pin. Extend the player character pin and get a reference to self.

Open ItemParent and in the viewport add a static mesh so that we can test it as well as have a default object. Drag it onto the root so that we can add physics. Check the enable physics box.

Go to the event graph and extend the Event Interact with’s Player Character pin and promote to a variable called Player. To check if the item is already being held, go to the First/Third person character BP and add a new variable of actor type called CurrentHeldItem. Extend the player output pin and add a get current held item node. Right click it and press convert to validated get.

Make 2 new functions called Pick up Item and Inspect item. Extend the Is Not Valid pin and add the Pick Up Item node and extend the Is Valid pin and add the Inspect Item node. Go to the Pick Up Item fn and extend it and add an Attach to component node. Add a reference to the Player variable, extend it and add a get ItemGrip node and extend it and hook it to the Parent pin of the AttachToComponent node. Set the Location Rule to Snap to Target and Rotation Rule to Snap to target so that the object moves to the position. Extend the output exec pin of the AttachToComponent node and add a Set Simulate Physics, set it to false and set the target as static mesh and extend the output exec pin and add a Set Collision Enabled node and set target to static mesh and the collision type to no collision. Extend its output exec pin and add a set Currently Held Item node and to the currently held item pin add a reference to self and extend the target and add a get player variable node.

We disable physics and collisions otherwise the object might intersect with player mesh and the resulting collision would disable all movement.

Now test this and whenever you press E while looking at and interactive item it gets attaches to the ItemGrip.

**Drop Item**

Open the First/Third person character BP and in the event graph add the Cancel node and add a reference to CurrentlyHeldItem. Right click it and select validated get option. Hook its input exec pin to the pressed pin. Change the currently held item variable type to item parent object as you’re only going to be holding objects with root as item parent, we also need to do this as we are going to use some functions from the Item Parent.

Go to item parent and add a new function called Drop Item. Extend it and add a detach from actor node and set Location, Rotation and Scale Rule to Keep World. Extend it and add a line trace by channel node. Add the get camera manager node, extend it and add the get camera location and hook it to start pin. Extend the camera manager node and add a get camera rotation node. Extend it and add a get forward vector node, extend it and add a vector\*int node and set the other pin value to 200. Extend it and add a vector+vector node and to the other pin hook the return value of the Get Camera Location node. Extend the output pin and hook it to end pin. Set the trace channel to Camera. With this we have the location of a point which is at a certain distance from the camera. We detach the object and now we have the location as to where to put it.

Split the out hit pin and extend the out hit location and add the set actor location node. Hook it’s input exec pin with the line trace by channel node’s output exec pin.

Extend it and add the set simulate physics node and set it to true and the target to the static mesh. Extend it and add the set collision enabled and set the target to the static mesh and set the type to Collision Enabled (Query and Physics).

Go to the first/third person character blueprint and in the Cancel node section extend the validated get’s currently held item pin and add the Drop Item fn node and hook its input exec pin with the is valid pin. Extend the output pin and add the set Currently Held Item node but no value is given so it becomes null.

**Inspect item**

To implement inspection first we need to disable the player controller’s control over the character and transfer that control over the object. Next, we need to change the position of the object. Then we need to be able to rotate, zoom and finally exit the inspect mode.

We have an inspect item fn in ItemParent already go to it. Extend the pin and add a disable input node. Add in a reference to the player variable and hook it to Target pin. Extend the Player Controller and add the get player controller node. Extend the output exec pin and add the enable input node. The target is self and hook the get player controller to the player controller pin.

To change the position, extend the output exec pin of the enable input node and add a set actor location node. Add a get player camera manager node and extend it and add a get camera location node. Extend the get player camera manager node again and add a get camera rotation node and extend it and add a get forward vector node and extend it and add a vector\*int node and set the other value as 200. Extend it and add a vector+vector node and to the other pin hook the output pin of the set camera in location node. Hook the output pin to the new location pin of the set actor location node.

To test this much go back to the first/third person character BP and in the event graph when we press E if it’s valid, we pick the item but if LookAtActor is invalid we have to interact with the currently held item. So, extend the is not valid pin and add the interact with node and extend the target and add the CurrentlyHeldItem, extend the player and add a reference to self.

Now if you test this the moment you press E the object appears at the screen centre and the player won’t be able to move. We need to be able to rotate it next. The logic for that is first check whether the left mouse button is pressed and if yes then take the axis value and multiply it and then add it with the actor’s current rotation and finally set that as the new rotation.

Go to the event graph of ItemParent and make a new variable called isRotating of Boolean type, add a left mouse button node and extend pressed and add a set isRotating node to true. Extend released and add a set isRotating node to false.

Add a mouse X node and extend the exec pin and add a branch node. To the condition add a get isRotating node. Extend the axis value pin and add a float\*float node and set the other value to -5. Add a combine rotators node and split the second pin and hook the output of the float\*float node to the yaw pin. To the first value pin of the combine rotators node add a get actor rotation node. Extend the combine rotators node’s output exec pin and add a set actor rotation node. Hook its input pin to the true pin of the branch node.

Now if you test this you will be able to rotate the object in the x direction. We need to rotate it about the y direction too.

Add a mouse Y node and extend the exec pin and add a branch node. To the condition add a get isRotating node. Extend the axis value pin and add a float\*float node and set the other value to -5. Add a combine rotators node and split the second input pin and hook the output of the float\*float node to the pitch pin. Extend the first input pin and add a get actor rotation node. Extend the output pin of the combine rotators node and add a set actor rotation node and hook its input exec pin to the true pin of the branch node.

Next, we need to be able to zoom in and out. For that we set a zoom level, we set a range for the zoom level. With every scroll up the level increases by one and vice versa. With each increase in zoom level the object is brought closer to you by some distance.

Make a new variable called ZoomLevel of int type. Add a mouse wheel up node. Add a get ZoomLevel node, extend it and add an int+int node to the other pin set the value as 1. Extend it and add a clamp(integer) node with min as 1 and max as 20. Extend the return value and add a set Zoom Level node. Hook its input exec pin to pressed pin of the mouse up node. Extend the value and add an int<=int node and set the other value as 20. Extend it and add a branch node and hook the branch node’s input exec pin with the set zoom level node’s output exec pin.

Add a get player camera manager node and extend it and add a get camera location node. Extend the player camera manager node again and add a get camera rotation node. Extend it and add a get forward vector node. Extend it and add a vector\*int node. Extend the other input pin and add an int\*int node. To one pin add a get zoom level node and set the other value as 10. Extend the output pin and add a vector+vector node. To the other pin hook the return value of the camera location node. Extend the output and add a set actor location node. Hook the input exec pin with the true pin of the branch node.

add a mouse wheel down node. Add a get zoom level node. Extend it and add an int – int node. Set the other pin as 1. Extend it and add a clamp(integer) node with min as 1 and max as 20. Extend the return value and add a set zoom level node. Hook its input exec pin with the pressed pin of the mouse down node. Extend the value and add an int>=int node and set the other pin to 1. Extend it and add a branch node and hook the branch node’s input exec pin with the set zoom level node’s output exec pin.

Add a get player camera manager node and extend it and add a get camera location node. Extend the player camera manager node again and add a get camera rotation node. Extend it and add a get forward vector node. Extend it and add a vector\*int node. Extend the other pin and add an int\*int node. To one pin add a get zoom level node and set the other value as 10. Extend the output pin and add a vector+vector node. To the other pin hook the return value of the camera location node. Extend the output and add a set actor location node. Hook the input exec pin with the true pin of the branch node.

Now we must add the functionality to return back to the hold item state. So we need to disable the controller action on the object and the controller must act on the player.

When you press E the interact fn, if we are already holding the object, calls the inspect fn. The inspect fn passes player controller from the character to item parent which means we can use inputs in item parent so to call the escape interact fn we can add the input functionality in the item parent itself because when we call it input will be enabled for the item parent class.

In the ItemParent BP create a new fn called stop inspect. Add an input action interact node. Extend the stop inspect node and add a disable input node. Extend the player controller and hook it to get player controller. Extend the output exec pin and add an enable input node. Extend the player pin and add a get player variable node. Extend the player controller pin and add a get player controller node. Extend the output exec pin and add a set relative location node and set the location as 0,0,0 and extend the exec pin and add a set relative rotation node and set the rotation as 0,0,0.

Go back to the event graph. Extend the pressed pin of the interact node and add the stop inspect fn node.

**Message Box**

We need a message box which appears when we pick up an item.

Make a widget BP called ItemHUD. It is the main HUD in this tutorial. Insert a border and inside it a text box such that the border is the parent of the text box. Anchor it. Give the border some color and padding and set both the widget’s size to content to true. Also enable alignment according to the position.

Set the text box name as MessageText and set the is variable to true. Set the border name as MessageBox and set it as a variable. This is because this HUD is going to be the main HUD and we need to show and hide it accordingly.

Go to First/Third person game mode BP and add an event begin play node. Extend it and add a create widget node and set the BP as ItemHUD. Extend it and promote it to a variable called ItemHUDVariable, extend it and add the add to viewport node.

Now if you test it on begin play you will see the text and border on screen. Go to the ItemHUD event graph and make a new fn called Update Message and add an input to the fn called Text To Display. The logic is to check if it has a value if yes we show that on screen else we hide the text and border. Extend the Text to Display pin and add an == node. Extend the other input pin and add a make literal text node and leave the value as blank. Extend the output pin of the == node and add a branch node. Hook its input exec pin to the Updata Message node’s output exec pin. Add a reference to Message Text variable and add a Set Text node. To the value hook the Text to Display. Hook the input exec pin of the set text node with the false pin. Add a reference to the Message Box variable, extend it and add a Set Visibility node and set the is Visibility to Visible. Hook the input exec pin to the output exec pin of the Set Text node.

Add an output to the UpdateMessage fn called isVisible of Boolean type. Extend the output exec pin of the set visibility node and add a return node and set the isVisible value to true.

If the text is null we have 2 possibilities – the player doesn’t have an object in his hands or he has dropped it or the player has an object already in his hands. If he doesn’t have an object the box and text must be hidden. If he has then the message must include the item name and further options available to him.

Go to the event graph of ItemHUD and near the event construct add a get player character node. Extend it and add a cast to first/third person character node. Extend the return value and promote it to a new variable called Player Character. Hook its input exec pin to the pin of the event construct node.

Go back to the update message fn and add a reference to the Player Character variable. Extend it and add a get CurrentlyHeldItem node. Right click it and select the validated get option. Extend its input exec pin and hook it to the true pin. Add a Message Box variable and extend it and add a set visibility node. Set the is visibility to hidden. Extend the Is not Valid pin and add hook it to the input exec pin of the set visibility node. Extend the output exec pin and add a return node and set the is visible value to false.

Add a Message Text variable reference extend it and add a set text node and hook it’s input exec pin to the is valid pin. Add a reference to the Message Box and extend it and add a set visibility node and set the visibility to visible. Extend the Input text pin of set text node and add a format text node and enter the input as –

Press E to inspect {name}

Press R to drop

To make a new line press Shift + Enter. To get the name of the item we need to store a name in each item’s BP. Go to the Item Parent BP and make 2 variables of string type – item name with default value as Item Parent and item description.

Go back to ItemHUD. Extend the Currently Held Item pin from the validated get node and add a get item name node and hook the value to the name pin of the format text node. Add a MessageBox variable and extend it and add a set visibility node and set the visibility as Visible. Extend the output exec pin of the set text node and hook it to the set visibility node’s input exec pin. Extend the output exec pin of the set visibility node and add a return node and set the is visible value to true.

Go back to the event graph, extend the set Player Character node in the event construct region and add an update message fn node and leave the input values as blank. Now we also need a message about the item when we look at it.

Go to the first/third person character and add an event begin play node. Add a get game mode node. Extend it and add a cast to first/third person game mode node and hook its input exec pin with the event begin play node’s output exec pin. Extend the as first/third person game mode pin and add a get ItemHUDVariable node and extend it and promote it to a variable called ItemHUDVar.

Go to the ItemParent node and in the event look at node extend the player character and add a get ItemHUDVar and extend it and add and update message node. Extend the text to display pin and add a format text node. In it type the text as –

Press E to pick up {name}

Extend the name pin and add a get item name node. Now if you test this the message is displayed once you look at an item but if you look away it still remains on screen.

Go to the First/third person character and go to the check look at fn and in the false pin section drag the return node away from the set look at actor node and drag in a reference of the ItemHUDVar and extend it and add an Update Message node and hook it between the set look at actor node and the return node. Leave the Text to Display as null. So now when we are not looking we don’t see the message.

Inspection UI

This contains the controls for the player to see what buttons can do what with the item and also the item description.

Make a new widget BP called InspectionScreen. Open it and add a border and anchor it to a good position. Add a color and padding. Add a grid panel inside the border and inside the panel add a text box. Set the text value as Press R: . Drag in another text box such that the 2 are side by side and this value will be Leave Inspection. Add another text box such that it’s below the Press R text and set its value as Mouse Wheel: . add another text box below the inspection and its value will be Zoom In/Out. Add another text box below wheel and its value will be Left Click & Drag: and another text box below zoom whose value will be Rotate. Set the border’s size to content to true. Align the boxes and adjust their paddings until you’re satisfied.

With this we have the various options one has with them. Next we add the item description.

Add another border and anchor it to a suitable location. Set the color. In it add a scroll box and name it ScrollBox and set it as a variable. A scroll box controls the text inside the widget. The reason I do this is to give the item description border the same width as that of the options box. Set the border’s width the same as the options box the length can be a lot more that the other (whatever looks fancy). Add a text box into the scroll box and name it ItemDescriptionText and set it as a variable. The scroll box automatically adds in the scroll tools in case the item description is very large.

Make a new variable called ItemDescription of text type. In the event graph add an event construct node. Add a reference of the ItemDescriptionText and extend it and add a set text node. Extend the input text pin and add a get ItemDescription node.

Add an event tick node. Add a reference to scroll box, extend it and add a set scroll offset node. Add another reference to scroll box, extend it and add a get scroll offset node extend it and add a float+float node and to the other pin set 1 for now and hook the output into the new scroll offset pin.

We want to delay the start of this as this is to occur only when scroll starts and scroll must not be allowed to exceed a certain value and at the end of the scroll the scroll value must be set to the initial value.

To do this make a new event called Start scroll. Extend it and make a gate node. The pin must go into the start pin of the gate node. Hook the exec pin of event tick into the enter pin and hook the exit pin into the input exec pin of the set scroll offset node. Now scroll begins only if the scroll start event is called. To clamp incrementation of scroll extend the get scroll box and add a get scroll offset of end node and extend the value and add a normalize of range node. The scroll offset of end node’s output pin should be hooked to the max pin and the result of the float+float node must be hooked into the start pin. Extend the return value and add a float >= float node and set the other pin to 1. Extend the Boolean value and add a branch node and hook its input exec pin to the output exec pin of the set scroll offset node.

Make a new custom event called Stop Scroll and hook it to the close pin of the gate node. Extend the true pin of the branch node and add a Stop Scroll node. All we need to do now is to reset the scroll offset after some amount of time. To do that in the event construct region extend the set text node and add a set timer by event node. Hook the Boolean pin of the Start Scroll node into the event pin of the set timer by event node and also set the time to 2 seconds. So after 2 seconds start scroll will be initiated.

Extend the Custom Stop Scroll event node (the one attached to the gate) and add a set timer by event node and hook the Boolean pin of the Start Scroll node to the event pin. Also set the time to 2 seconds. It’s this node’s output pin that’s now connected to the close pin.

Add in another reference to the scroll box and extend it and add a set scroll offset node and set the value to 0. Hook the exec pin of the Start Scroll node to the input exec pin of the set scroll offset node and hook its output exec pin into the enter pin of the gate.

This logic creates a loop such that when the widget is created the scroll goes down, then resets after 2secs and then goes down again.

Go to the item parent BP and open the Start inspect fn. Extend the start inspect fn and add a create widget node and set the class to InspectionScreen and extend the item description pin and add a get item description node. Extend the output exec pin and promote it to a variable called InspectionUI. Extend the return value and add an add to viewport node. Hook this to the rest of the code.

But then the other HUD is still there and we need to remove that too at the start. So add a reference to the player, extend it and add a get ItemHUDVar node and extend it and add a set visibility node and set the visibility to hidden. Insert the node between the start inspect node and the create widget node.

If you test this, you’ll find that only the first line of the item description’s seen. Go to the ItemDescriptionUI BP and select the text box and either select auto wrap content but it’s best to enter a value. Eg – if border with is 580, the wrap text at value should be (580-30) and add padding as 30.

Now if you test it you get the description scrolling down and resetting every 2 secs. But when you leave inspection the HUD isn’t changing.

Open the Item Parent BP. Go to the Stop Inspect fn. At the end of the functionality and a reference to InspectionUI variable and extend it and add a remove from parent node. Hook it to the end of the code. Add a reference to the player variable and add a reference to the ItemHUDVar and extend it and add a set visibility node and set visibility to visible. Hook the output exec pin of the remove from parent node to its input exec pin.

Door and Key Example

To inherit all features from the ItemParent just right click on it and select create child BP and give the name as Key. Open it and set the static mesh as the item you want to set the mesh as. Set the item name and item description.

The logic is to check when we approach the door if we have the key if no tell player to get the key or else tell him to push a button to open the door. On button press open the door.

Make an actor BP called door. Add a static mesh and place it in the world. Open the BP and go to class settings and add the Interact\_interface. Make a variable called Key of type item parent and check the instance editable box. In the event graph add an event look at node, extend the player pin and add a get currently held item node. Extend it and add a == node and to the other pin add a get key node. Extend the output and add a branch node and hook its input exec pin with the output exec pin of the event look at node. Extend the player pin and add a get ItemHUDVar node. Extend it and add an update message node and hook its input exec pin to the true pin. Extend the Text to Display pin and add a format text node and in it type –

Press E to use {name}

Extend the currently held item node and add a get item name node and hook it to the name pin. Extend the ItemHUDVar pin again and add an Update Message node and hook it to false pin. Set the text to display field as –

Find key to unlock the door

Now we have to set a value to key. Click on the door in the scene and in the manager under default you see that the key variable is empty. Use the dropper pin next to it and click on the key in the scene. This sets the key variable to the key in the scene.

Now if you test this whenever you look at the door you see the find key message and once you have the key you find the press E message.

To open the door go to the door BP and add an interact with node. Extend the player character and add a get currently held item node and extend it and add an == node. Extend the other pin and add a get key variable node. Extend the output and add a branch node. Make a custom event node called Open Door. Extend the true pin and add an open door node. Extend it and add a destroy actor node and set the target actor to the key variable.

Extend the open door event node and add a timeline node and name it door position. Open it and set the positions such that it opens the way you want it to.

Now once E is pressed, the door opens and the key is destroyed too.