Jet Plane Tutorial

This is a tutorial to make a simple jet plane. Not based on advanced physics or anything but just simple math.

Import the plane model and it’s better if you have the flaps as separate meshes as we can change their rotations more easily using static meshes instead of using animations or skeletal meshes. Make the material and apply it to the mesh. For creating the jet glass material, you could set the lighting mode to surface translucency volume which would give the material an opacity parameter while keeping the specular parameter. The specular can be set to a constant and the opacity set accordingly and maybe set the roughness to 0 if that’s what you want.

In the content folder create a new blueprint class of type pawn and name it BP\_Jet. Open it and add the static meshes for the fuselage and make it the parent, add the glass static meshe and make sure its parent is the fuselage. Add a spring arm component and then add a camera as the child of the springarm. The springarm should be the child of the parent. Set the springarm length to say 800 and offset to say 300 so that it’s a bit above the plane check the enable camera lag and enable rotation lag and set their values to say 10 & 5. This is to make it look like the plane is moving and the camera follows it. To attach the aileron use the sockets in the fuselage. Add the aileron static meshes and set the as the child of the fuselage then set their parent sockets to the respective fuselage sockets. Same goes for the elevators, rutters and the flaps.

Set the game mode as the flying pawn and the pawn as the BP\_Jet. This step might change according to the game mode you want to use. Or just drag in the jet into the scene and set the auto possess player to Player 0.

Now on simulation you’ll see through the camera and but the plane won’t move. Next, we are going to set up the controls for the plane.

In the BP\_Jet event graph make a float variable called MaxThrustSpeed and set its default value to 10000 and set it to private by checking the Blueprint Read Only option and in the category type in Constants because we are going to consider such variables as constants to be used for simulation. Make another float variable called MinThrustToNotFall 4000 and make it private and set the category to Constants and similarity make ThrustMultiplier to 2500, Gravity to 981 and Drag to 0.25. As they are constants, we can’t set them during the game.

Create a float variable called MaxFlapPitch and set it to 10 and set it to private and in the category we want to show that although it’s a constant it belongs to a subcategory called ControlSurfaces which is what I want to call the variables that let you control the surface parts of the jet. To do that first assign the Constants category to the variable. Then edit the name by adding a | next to it and then typing the category – Constants|ControlSurfaces. Now the MaxFlapPitch is shown under the ControlSurfaces subcategory in the Constants category. Similarly make MaxElevatorPitch to 25, MaxRutterYaw to 45 and MaxAileronPitch to 45. We don’t need min variables as the min values are the negative of the max values.

Create a float variable called thrust speed, it’s NOT private, and set the category to DynamicVariables to indicate that they can change. Set its default value to 0. Similarly create CurrentSpeed and AppliedGravity. Thrust speed is how much force is applied to the jet and the current speed is the speed of the jet and the applied gravity is the result after factoring in the upward thrust.

Create a float variable called TargetYaw, it’s NOT private and set the category to DynamicVariables|Rotation. Similarly, make CurrentYaw, TargetPitch, CurrentPitch, TargetRoll, CurrentRoll. The target yaw is the target we want to reach to and the current yaw is the current yaw of the jet.

Make a function called Update Position which takes a float input called delta seconds and set the category to Private. Make a function called Update Yaw which takes 2 float inputs – delta seconds and yaw and set the category to Private. Similarly make Update Pitch and Update Roll. Make a function called PrintVariables solely for debugging.

Near the event begin play node, add a reference to MinThrustToNotFall node, extend it and add a set Thrust Speed node and extend the return value of the thrust speed node and add a set CurrentSpeed node. Hook the input exec pin of the ThrustSpeed node to the output exec pin of the begin play node.

Extend the event tick node and add the UpdatePositon function and hook the delta seconds of the event tick node to the delta seconds parameter of the UpdatePosition node. Extend the output exec pin of the UpdatePosition node and add the PrintVariables node. In the PrintVariables function definition, extend its output exec pin and add a sequence node. extend the then 0 pin and add a print string node and extend the input string and add an append node and to the A pin set Thrust and to the B pin hook the return value of the get ThrustSpeed node and hook the input exec pin of the print string node. Similarly print out the CurrentSpeed and AppliedGravity values. If you set the duration of the print string to 0 it’ll only 1 line will be seen on the screen.

Open the UpdatePosition function and extend the output exec pin and add a set CurrentSpeed node. If you turn on the thrusters, a jet is going to speed up right away but if you them off the speed doesn’t decrease right away, it does so over time. So if the thrust speed is less than the current speed we want the speed to decrease gradually but if the thrust speed is greater than the current speed then the current speed should be equal to it right away. Add a get ThrustSpeed node, extend it and add a float < float node and to the other pin add a CurrentSpeed node. Extend the result pin and add a select node and make sure the result is added to the index pin of the select node. To the false pin add a get ThrustSpeed node. Extend the true pin and add an FInterp to node. Extend the current pin and add a get CurrentSpeed node, extend the target pin and add a get ThrustSpeed node and to the delta time add a get DeltaSeconds node and to the InterpSpeed add a get Drag node as we use it to simulate air resistance. Hook the return value of the select node to the input pin of the Set CurrentSpeed node.

Now that we know the current speed we can calculate the new position. To do that add a get CurrentSpeed node and extend it and add a float \* float node. Extend the other pin and add a get DeltaSeconds node and extend the return value and add a float \* vector node. Extend the vector pin and add a get actor Forward Vector node. Extend the return value of the float \* vector node and promote it to a new variable called New Position. Hook the output exec pin of the set CurrentSpeed node to the input exec pin of the set NewPosition node.

In this simulation I’m only going to apply the effects of gravity when the plane slows down. Add a get CurrentSpeed node and extend the return value and add a map range clamped node. extend the in range B pin and add a MinThrustToNotFall node and extend the out range A pin and add a gravity node. This maps the value so if the value is equal to the InRange B then it outputs the OutRange B but if the value is equal to the InRange A then it outputs the OutRange A else if the value is anywhere between InRangeA & InRangeB then the output is mapped accordingly between OutRange A and OutRange B. So here, if the CurrentSpeed is 0 then the full effect of gravity is applied but if it’s equal to the MinThrustToNotFall then the gravity is 0 but if it’s anywhere between the 2 values then the gravity is mapped accordingly. Extend the return value of the map range clamped node and add a set applied gravity node and hook its input exec pin to the output exec pin of the set NewPosition node.

Next we update the position to factor in gravity. Add a get NewPosition node and extend it and add a break vector node. Add a make vector node and hook the X value to the X value and the Y value to the Y value. Extend the Z value and add a float – float node. Add a get AppliedGravity node, extend it and add a float \* float node, to the other pin set the DeltaSeconds node. Extend the return value of the float – float node and hook it to the Z pin of the make vector node. Extend the return value of the make vector node and add an AddActorWorldOffset node and make sure it’s hooked to the delta location pin. Hook the input exec pin of the AddActorWorldOffset node to the output exec pin of the Set Applied Gravity node. Check the Sweep pin as this ensures that the actor moves from the position to the next gradually thereby ensuring that it doesn’t pass through objects. (Not sure about this.)

Now on simulation the plane moves forward although we won’t have any control over it.

Add the input keys. Add axis mappings – one for thrust (W set the scale as 1,S set the scale as -1), yaw(D set the scale as 1,A set the scale as -1), pitch(Mouse Y set scale as -1 although some people like it at 1) and roll(Mouse X set scale as 1).

Go to the BP\_Jet event graph. Add the event Thrust. Extend the Axis Value and add a float \* float node. Click the + pin so that there are 3 float input pins. To the 2nd on add the get world delta seconds node and to the other pin add the thrust multiplier node. Extend the return value and add a float + float node and to the 2nd pin add the Thrust Speed node. Extend the return value and add a clamp node and set the min to 0 and extend the max pin and add a get MaxThrustSpeed node. Extend the return value of the clamp node and add a set Thrust Speed node. Hook the output exec pin of the InputAxis Thrust node to the input exec pin of the set ThrustSpeed node.

Now on simulation, if you hold the W or S the speed increases or decreases respectively. Also if the speed slows down below 4000 then the gravity’s effects are visible.

Add the InputAxis Yaw node and extend the output exec pin and add the Update Yaw node. extend the Axis value pin and hook it to the Yaw pin of the Update Yaw function. Extend the delta seconds pin and add a get world delta seconds node. open the Update Yaw function and extend the Yaw pin and add a set Target Yaw node. Add an Finterp to node. Extend the current pin and add a get CurrentYaw node. Extend the Target pin and add a get TargetYaw node. Extend the delta time pin and add a get DeltaSeconds node. Set the Interp speed to say 10. Extend the return value and add a set Current Yaw node and hook its input exec pin to the output exec pin of the set TargetYaw node. Extend the return value of the set CurrentYaw pin and add a float \* float node and click the + button so that there are 3 inputs. Extend the 2nd pin and add a get DeltaSeconds node and set the 3rd value to say 20. Extend the output exec pin of the set CurrentYaw node and add an AddActorLocalRotation node and extend the return value of the float \* float node and hook it to the delta rotation Z pin of the AddActorLocalRotation node.

Now on simulation when you press A or D the jet moves to the left or right.

Now we also want to rotate the rutters according to the Yaw. The rutter yaw should be mapped between the max. rutter yaw and the min. rutter yaw depending on the current yaw. To do that after the AddActorLocalRotation node add a reference to the RuttlerL & the RutterR nodes. Extend the RutterL node and add a SetRelativeRotation node. add a Map Ranged Clampled node and to the Value pin hook the CurrentYaw node, to the In Range A value set -1 and to the In Range B value set -1 and to the OutRangeA value hook the MaxRutterYaw node and extend the OutRangeB value and add a float \* float node and to the 1st pin hook the MaxRutterYaw and to the 2nd pin set -1. To make the code cleaner you can make a macro called negate float and add a float \* float node and hook the A pin into the Input node and name the input as in and set the B pin to -1 and hook the output pin to the Output node and name the output as out. Split the struct pin of the rotation in the SetRelativeRotation node and hook the return value of the Map Range Clamped node to the Yaw pin of the SetRelativeRotation node. Do the same thing for Rutter R (just hook the return value of the map range clamped to its yaw too.)

Now on simulation the rutter rotation changes accordingly.

Add an InputAxis Pitch node and extend its exec pin and add an UpdatePitch node. Hook its axis value to the pitch pin and extend the delta seconds pin and add a get world delta seconds node. Open the UpdatePitch function copy the functionality of the UpdateYaw function and paste it in the UpdatePitch and just replace every yaw with pitch equivalent. If you want to do it from scratch then – extend the pitch pin and add a set TargetPitch node. Extend its output exec pin and add a set CurrentPitch node. Add an FInterp to node, extend the Current pin and add the CurrentPitch node, extend the Target pin and add the TargetPitch node, extend the delta time node and add the Delta Seconds node and set the interp speed to say 10. Hook its return value to the input pin of the set CurrentPitch node. extend the return value of the set CurrentPitch node and add a float \* float node and click the + button so that there are 3 pins. Extend the 2nd pin and add a set DeltaSeconds node and set the 3rd value to say 20. Extend the output exec pin of the set CurrentPitch node and add an AddActorLocalRotation node and split the struct pin of the rotation and hook the value of the float \* float node to the Pitch pin. Add a reference to the elevator L and extend it and add a SetRelativeRotation node. Add a Map Range Clamped node and extend the value pin and add a CurrentPitch node and set the In Range A to -1 and In Range B to 1 and extend the OutRange A and add a get MaxElevatorPitch node and add a get MaxElevatorPitch node and extend it and add a Negate Float macro which was created earlier and hook the out pin to the OutRangeB pin. Hook the return value of the Map Range Clamped node to the Pitch pin of the SetRelativeRotation node. Add the elevator R and extend it and add a SetRelativeRotation node and hook the return value of the Map Range Clamped pin to the Pitch pin of the SetRelatvieRotation node. Do the same for the Flaps too.

The Roll is a little different because the ailerons are not going to move the same way. Add the InputAxis Roll node and extend the exec pin and the Update Roll node. Extend the Delta Seconds pin and add a Get World Delta Seconds node. extend the axis value and hook it to the roll pin. Open the Update Roll function. Except for the part rotation control the functionality is the same so just copy that and then on simulation the ailerons move in the same way but we want it to move in opposite directions. To get that done for the left aileron just swap the OutRange A and the OutRange B so the negated Max Aileron pitch goes in A and the MaxAileronPitch goes in B. (Although it’s roll here remember to set the Aileron rotation to Pitch and not roll.)

Now on simulation the Aileron goes in opposite ways when we roll.