GameDev Diaries

Making a First-Person Shooter Game

FPS

Made a 3rd person project FPS as the Unreal First-Person template showed only the user hands and gun which wasn’t as cool as wanted.

**Adjusting camera**

Opened the third person character blueprint and in the viewport moved the camera to the player head but on playing as the head moved forward first, the head was visible. To change that, made the follow camera object as the child of the mesh and set head as the parent socket of the camera. Adjusted the camera rotation to -90,0,90 and also adjusted the location. But then the camera wouldn’t respond to mouse rotation so checked the use pawn control rotation option. Selected the third person character option and checked the use controller rotation yaw option. Now the camera moved in sync with the head and on looking down the rest of the body was visible.

**Player Health and Armour**

Opened third person character blueprint and made 2 variables of float type – Health and Armour and set both their default values as 1.0. To display them on the screen made a widget class and named it HealthHUD. Inserted 2 progress bars and placed them on the top right corner of the screen and also adjusted their size and anchored them to the top right hand corners of the screen and set the Health progress bar color to light red and the Armour progress bar color to light orange and binded the Health progress bar and in the graph add a cast to third person character node and set the object wildcard as the get player character node and extended the as third person character and added a get Health node and plugged it into the return value. Did the same procedure with the Armour progress bar. To make them appear on the screen, opened the third person character blueprint event graph and added a begin event node and extended it and added a create widget node. Set the class as HealthHUD and extended it and added an add to viewport node. As the default values are 1.0, on runtime both the bars are coloured completely.

**Damage and armour regeneration**

We want the armour to regenerate but not the health. Armour regeneration logic is this: at every event tick after a delay, we need to check if the armour is less than 1. If it is less then 1 it has to be increased by 0.01 i.e. 1% of its present value. To regenerate the armour, in the third person character blueprint add an event tick node. Extend it and add a delay node and set the delay value as 1 second. Extend it and add a sequence node (we add a sequence node for adding an extra functionality later) and extend the Then0 pin and add a branch node. Extend the condition and add a float<float node and to one input add the get Aromour node and set the other input as 1. Extend the true and add the set Armour node and extend its input and add the float+float node and to 1 input add the get Armour node and set the other input as 0.01.

For the take damage part, the logic is this: we need to decrease the armour by 0.05 i.e. 5% and then we need to check is the armour is less than 0. If it’s true we need to decrement the value of health by adding the current health with the armour value and in the end just set the armour value as 0. This logic is linear and easily executable and removes the need for any extra branching.

Made a TakeDamage5% function and opened the graph. Extended it and added a set Armour node and extended the input value and insert a float + float node to one input added a get Armour node and set the value of the other as 0.05. Extended the exec pin of the set Armour node and added a branch node. Extended the condition pin and added a float<float node and to one pin added the get Armour node and to the other set the value as 0.0. extended the true pin and added the set Health node and extended the Health value and added a float+float node and to one input added the get armour node and to the other added the get Health node. Extended the exec pin of the set Health node and added a set Armour node and set its value as 0.

To test the functionality made a new actor class and named it damager. Added a box collision in it and adjusted its dimensions. Dragged it into the scene. In the event graph extended the event actor begin overlap node and added a cast to third person character and set the object wildcard as get player character. To know when the character is in the box and when he isn’t made an isInBox variable of Boolean type and extended the exec pin and made a set isInBox node and checked it so as to set it to true. Extended the as third person character and added the TakeDamage5% node and hooked its input exec pin to the set isInBox’s exec pin. Extended the function’s output exec pin and added a branch node and set the condition as the get isInBox node and added a Delay node and set the duration to the appropriate time and hooked its exec pin back to the damage function so as to form a loop. Added an event actor end overlap node and extended it and added a set isInBox node and unchecked the box so as to set the variable to false.

The delay node mentioned above is important as while testing without it an infinite loop error was thrown because when the actor steps in the box, due to the lack of the delay node the control goes back to the function and again and again making an infinite loop. With the delay node the player has time to go out and set the Boolean variable as false.

**Blood Splash Effect**

When the player experiences damage, a red vignette effect on the screen would look cool. What is needed is a red vignette to appear and fade when damage occurs. So, downloaded a red vignette image. Although a photoshopped image would be excellent, this is okay. Created a widget and named it RedVignette. Inserted an image in the designer and set position as 0,0 and size as 1920,1080 and anchored it to the frame and in the appearance, set the image as the one which was downloaded and set its opacity as 0. Named the image as Blood and created an animation out of the image (Not the image panel) and named it DamageEffect and set its track as opacity and at 0.5s set the alpha such that the red was visible but the background wasn’t and at 1s set the opacity back to 0. We want this to appear as soon as the widget is displayed i.e. as soon as it’s constructed. So, in the event graph extended the event construct and added a play animation node. Added a reference to the DamageEffect and hooked it to the in-animation input.

To call this animation went to the Take Damage 5% function and to the end of it added a create widget node and set the class as RedVignette and the owning player as the get Player Controller node and extended the return value and added the add to viewport node. On testing animation is not called immediately because at the beginning the armour is max and we added the above functionality to the part when Armour is 0 i.e., we added this after the branch true part. We need to call this even when Armour isn’t equal to 0 i.e. to the false part of the branch statement. So hooked the false pin to the create widget node an after that it got called whenever damage took place.

**Setting up Character Animations**

Downloaded the Virtus resources to get all the animations for the character. Made a new folder named Character. Imported the Swat mesh without any skeleton but with the materials. Opened it and tested the movement of all the body parts. Next made a new folder named animation and in it imported all the animations and in the dialog box made sure that the mesh was set as the Swat skeleton and then imported all. In its parent folder made an animation blueprint and named it Walk\_Run\_BP. As the only 1 animation can be plugged in at a time instead of using animations made a state machine node and named it as Walk\_Run and dragged it into the final animation pose and compiled. Double clicked the state machine node and in the anim graph extended the entry and added a state and named it idle and extended it and added another state and named it Walk\_Run. The arrow mark means that from idle it’s possible to go to the Walk\_Run state. So, extended the Walk\_Run state and pointed it back to Idle as it’s possible to go from Walk\_Run state back to idle.

Double clicked the idle state and in its anim graph dragged in the idle animation and hooked it to the final animation. For the Walk\_Run state went back to the content browser and created a blend space animation and named it Walk\_Run\_BS. Opened it and we see a grid. Set the name of the horizontal and veritical axes as direction and speed respectively and set their min. and max. values as -180,180 and 0,600 (600 is the max. speed that can be measured by character animation). All we need to do now is to plot the animations according to the direction and speed.

As at speed 0 the player is idle in all the bottom 5 spots drag in the idle animation. At speed 300 the player is waking. At direction 0 he is walking forward, direction 90 he’s walking right, direction -90 left, direction 180 as well as -180 he’s walking backwards. At speed 600 he’s running in the respective directions. To preview the animations in the grid press Shift and drag the cursor and the transitions can be previewed.

The diagram in the resources give a clear idea on how to do the movement works with speed and direction.

**Finishing the animation blueprint**

Opened the Walk\_Run\_BP and clicked the Walk\_Run state and dragged in the Walk\_Run\_BS and hooked it to the final animation. To set the direction and speed extended both and promoted them to a local variable and set both their names as the Direction and speed respectively. In the event graph of the Walk\_Run\_BP extended the event blueprint update animation node and added a sequence node and extended the then0 pin and added a cast to character node and set the object wildcard as the Try Get Pawn Owner node and extended its return value and made a Get Velocity node and a Get Control Rotation node and added a calculate direction node and hooked the return value of the get velocity node to the velocity pin and the get control rotation node’s return value to the base rotation pin. Extended the cast to character node and made a set Direction node and hooked its value to the return value of the calculate direction node. Extended the set direction node’s exec pin and made a set speed node and to set its value extended the get velocity node and added a vector length squared node and then extended its output and hooked it to speed as speed is a scalar and velocity a vector.

To tell when the idle state and when the Walk\_Run states must be called, went to the anim graph and clicked on the idle to walk\_run transition rule and in its event graph extended the can enter transition pin and added a float>float node and to the first pin added a get speed node and to the second set the value as 10 as when speed becomes greater than 10 we want the player to walk and run. Went back to the anim graph and clicked on the walk\_run to idle transition rule and in its event graph extended the can enter transition pin and added a float<float node and to the first pin added a get speed node and to the second set the value as 10 as when the speed becomes less than 10, we want the player to stand idle.

Opened the third person character blueprint and selected the mesh and set skeletal mesh as Swat Mesh and to get the animation set the animation as the Walk\_Run\_BP. Readjusted the position and rotation of the camera. On playing, the speed and direction animation worked as planned but on releasing the forward key, the player comes to stop immediately and the screen stops like a dirty flicker. Instead of that to make the player movements blend into each other, opened the blend space i.e. Walk\_Run\_BS and set the interpolation time to 1. At first, no visible effect was seen that was because it had to be changed on both axis in this case.

**Adding an AK47**

Made a folder called weapons and a subfolder AK47. Imported the gun from the resources and checked the skeletal mesh while importing. Imported the diffuse, normal and specular maps and in the gun’s material blueprint hooked the textures to their respective slots. Created a new blueprint actor and named it Weapon\_Base (not AK47\_Base as this was intended to serve as templates for future guns) and added in the AK47 skeletal mesh in the viewport. Opened the swat mesh and clicked the skeletal mesh and in the left-hand window right-clicked the right hand (as that bone is preferable for guns) and created a socket and named it Weapons\_Attach. Right clicked it and in the preview mesh option, selected the AK47 and adjusted its transformation so as to get the desired effect.

To spawn the object in the player, opened the third person character blueprint and in the event graph extended the HealthHUD’s add to viewport node (as we need the gun to spawn at the beginning of the game and the HealthHUD functionality is connected to the on event begin node) and added a spawn object from class node and set the actor as Weapon\_Base class and extended the spawn transform pin and added a make transform node. We don’t need to set the values to anything. Just extend the return value of the Spawn Actor from class node and add an attach to component node. Brought in a Mesh reference and attached it to the Parent pin and set the socket to Weapon\_Attach.

On playing the weapon appears. Further adjustments in transformation could be made in the the character mesh’s socket preview. Some clipping was seen. To remove this went to project clipping and in the general settings, set clipping as 1.

**Firing the AK47**

Made a blueprint actor called Bullet and added a static mesh. Set the mesh as narrow capsule and adjusted the size of the capsule. Added a projection component and set the speed as 400 for debugging purposes and set gravity as 0.05 so the bullets don’t fall to the ground. To spawn the bullet, opened the AK47 static mesh and clicked the skeletal mesh. Added a new socket and named it muzzle. Altered the transform of the muzzle such that the socket is at the tip of the gun. In the event graph added a custom event node and named it Fire and extended it and added a Spawn actor from class node and set the class as Bullet. Dragged in a reference of the AK mesh and extended it and added a get socket transform node and set the socket name as Muzzle and hooked in the return value as Spawn Transform.

To trigger the function on mouse click, opened project settings and under the input open created a new binding and named it PrimaryFire and set the action as Left Mouse Button. In the third person actor class in the event graph added a Primary fire node. To call the fire function of the particular weapon its reference is needed. To get that, extended the SpawnActor Weapon Base node’s return value and promoted it to a local variable named current weapon and hooked the exec pins.

After the PrimaryFire node dragged in a Current Weapon reference and extended it and added a Fire node and hooked its exec input pin to the pressed pin.

On playing the bullet direction and position needed to be altered. The direction was altered by changing the rotation of the socket. The bullet position was made from horizontal to vertical by changing the narrow capsule transform.

**Crouching with animations**

To make the character crouch on pressing ctrl, went to project settings and in the input made a new binding named Crouch and entered Left Ctrl and Right Ctrl as input keys. Went to the content browser and in the character folder made a new blend space called Crouching\_BS. Opened it and set the axes as Direction and Speed and min. and max. values as -180,180 and 0,300 respectively. Set max. speed as 300 because while crouching, speed is less. At speed 0 set the animation as idle crouching at 300,0 walk\_crouch\_forwards, 300,90 walk\_crouch\_right, 300,-90 walk\_crouch\_left, 300,180 and 300,-180 walk\_crouch\_backwards.

Opened the Walk\_Run\_BP and in the anim graph from the idle state made an arrow and added a new state named Crouching and made an arrow back from crouching to idle. Made an arrow from Walk\_Run to Crouching and back. Double clicked the Crouching and in its anim graph dragged in a reference to the Crouch\_BS and hooked it to the final animation. Dragged in a reference of Direction and Speed and hooked them to their respective pins. We use the same variables so now we don’t need to calculate them separately. We can use the same values.

To trigger the state we need to know whether the character is crouching or not. Went to third person character and made a Boolean variable isHeCrouching as isCrouching or something similar to that seemed to be a default variable. As we had earlier made a Crouch input event, added a crouch node and extended the pressed and set isHeCrouching to true and extended the released and set isHeCrouching to false. We use the same speed variable but that variable can exceed 300. To prevent that from happening during crouching, dragged in a character movement reference and extended it and added a set max walk speed and hooked its input exec pin to the first set’s output exec pin and set max. speed as 300. To change max. speed to 600 while standing, did the same thing with the second set only this time set max. speed as 600.

Now we have a variable for crouching in the third person character but we need a variable of the same value in the Walk\_Run\_BP as it’s not optimal to cast to the third person character every time we want the value. So, to make a local variable in the Walk\_Run\_BP with the same value as isHeCrouching, opened the event graph of Walk\_Run\_BP and extended the Then1 pin and added a cast to third person character node and set object wildcard as try get pawn owner. Extended the as third person character pin and added a get isHeCrouching node and promoted it to a new variable called CrouchingOrNot. Hooked its input exec pin to the cast to third person character’s output exec pin.

Now that we have the variable we just need to check if its true or false to transition between other states. Clicked the idle to crouching condition and extended the result pin and added a Boolean==Boolean node and to one hooked the CrouchingOrNot variable and checked the other box. So, whenever it’s true the transition happens. Did the same for walk-run to crouching condition. Did the same for the other 2 transitions – crouching to idle and crouching to walk\_run only this time the other box was unchecked such that the transition happens only when CrouchingOrNot is false.

On run-time to prevent the sudden crouching movements, opened the Crouch\_BS and set the interpolation time to 1 on both axes but then when we stop movement it moves for a bit longer time especially because speed is low so the interpolation time has a larger effect so set it to 0.2.

**Sprinting with animation**

Opened the project settings and in the input section added a new binding named Sprint and set the input keys as Left and Right Shift. Made a new blend space in animation folder named Sprint\_BS and set the axes as Direction and Speed and the min. and max. values as -180,180 and 600,1200 respectively. For the 600 line set the animations as run\_backwards, run\_left, run\_forward, run\_right and run\_backwards and on the 1200 line set the animations as sprint\_backwards, sprint\_left, sprint\_forwards, sprint\_right, sprint\_backwards respectively.

Opened the third person character blueprint and to check whether Sprint is activated made a Boolean variable named isHeSprinting. Added a Sprint node and extended the pressed pin and set isHeSprinting to true and dragged in a character movement reference and extended it and added a set max walk speed node and set the value as 1200 as the current max speed was 600 and 1200 was needed to activate the sprint animations. Hooked the set max walk speed node’s input exec pin to the set isHesSprinting node’s output exec pin. Extended the release pin and set isHeSprinting to false and dragged in a character movement reference and extended it and added a set max walk speed node and set its value to 600 and hooked its input exec pin to the set isHeSprinting node’s output exec pin. This is to reset the speed to 600 or else just pressing the move keys for too long could accelerate the speed to 1200.

Opened the Walk\_Run\_BP and in the anim graph extended the Walk\_Run and added a new state named Sprinting and pulled an arrow back from Sprinting to Walk\_Run. As players can sprint only after a walk/run state transitions from the idle or crouching states were not included. As we need a local variable to check for sprinting. Opened the event graph and added a pin to the sequencer and extended Then2 and added a cast to third person character node and set the object wildcard as try get pawn owner node and extended the as third person character pin and added a get isHeSprinting node and extended the pin and promoted it to a local variable named Sprinting or not. Hooked the input exec pin of the variable node to the cast to third person character node’s output exec pin.

To set the transition conditions went back to the anim graph and double clicked the Walk\_Run to Sprinting condition and extended the can enter transition pin and added a Boolean==Boolean node and checked one box and to the other hooked a Get Sprinting or Not node. Did the same to the Sprinting to Walk\_Run condition only in this case unchecked the box so that it’s called whenever the variable is equal to false.

Also set interpolation of both axes in the BS to 1.

**Using control rotation**

To make the gun aim at wherever the mouse points to all we need to do is to make the character rotate along the y axis as the gun already points to whatever direction of the x and z axes the mouse is pointing in. The character we use has 3 spines so to make the gun point in the y direction all we need is to change the rotation of these 3 spines. Just 1 spine would do by still in this case we use 3 spines.

Opened the Walk\_Run\_BP and in the anim graph (the outer most one) extended the gap between the Walk\_Run state node and the final animation and in between them added 3 modify bone transformation nodes. In the right window unchecked the location, scale and alpha as we are only concerned with rotation of the bone. As we want the rotation to modify the current value, set the rotation mode of all 3 modes to Add to Existing. Set the bone to modify to Spine, Spine1 and Spine2 respectively. Extended the rotation input of one of the nodes and promoted it to a local variable named AimRotation and hooked its value to the other 2 nodes. Connected all the men shaped nodes to one another.

To get the value of AimRotation, went to the event graph of the Walk\_Run\_BP and added a new pin to the sequence node. Extended Then3 and added a cast to third person character node and set the object wildcard to try get pawn owner node. Extended the as third person character and added a get control rotation node and extended it and added a break rotator node as we are only concerned with the y value. Now the mouse can move in all directions but the player should anly be able to move the gun up and down and not in full cycle so we need to modify the control rotation value such that it doesn’t exceed the permissible limits. So extended the y value and added a float-float node and set the top value as 360 and hooked the pin to the bottom value. This is to set the rotation within 180 even if the value exceeds 180. If it exceeds 180 then it would mean that the character is going in a U shape backwards. Extended the y value again and added a float\*float node and set the other value as -1. Extended the y value again and added a float>float node and set the other value as 180. Extended its value pin and made a select float node and set A as the output of the float-float node and B as the output of the float\*float node. So if y is greater than 180 A is selected (360-value) else B (-value) is selected. Extended the output pin and added a set AimRotation node and hooked its input exec pin to the output exec pin of the cast to third person character node.

**Fully Automatic Rifle**

Right now, our rifle fires only once when the left-mouse button is clicked. We want the firing to be automatic and also want an ammo count.

In the weapons>AK47 folder opened the Weapon\_Base blueprint and made a variable ammo of integer type and set its default value to 30 as there are 30 bullets in an AK47 magazine. Made another variable FireRate of float type and set its default value as 0.1. This is the rate at which the rifle fires in 1 sec and by setting it to 0.1 it can fire 0.1\*10=1 i.e 10 bullets in a second.

Opened the event graph and made an integer-integer node. To one input pin added a get ammo node and to the other set the value as 1. Extended the output pin and added a set Ammo node and hooked its input exec pin to the spawn actor bullet’s output exec pin.

Opened the third person character blueprint and went to the firing part. Extended the gap between the Primary Fire node and the Fire function. Made a new Boolean variable named isHeFiring. Extended the pressed pin and added a set isHeFiring node and the value to true. Extended the exec pin and added a branch node. To the branch node’s condition pin added an integer>integer node and extended the Current Weapon reference and added a get Ammo node and hooked it to the top pin and set the bottom pin value as 0. Extended the true pin and hooked it into Fire. Extended the Fire pin’s exec pin and added a delay node. Extended the Current Weapon reference and added a get FireRate node and hooked it to the delay value. Extended the completed pin and added a branch node and to the condition added a get isHeFiring node. Extended the true output pin and plugged it into the first branch node’s exec pin. So that a loop is formed and at each iteration ammo value is ensured to be above 0. Now extend the released pin and added a set isHeFiring node and set the value to false.

To display the ammo value on screen opened the HealthHUD and added a text to the top right-hand corner and anchored it to the same spot and set default text as 0 and created a binding for it. In the event graph extended the get text and added a cast to third person character and set the object wildcard as get player character and extended the as third person character and added a get Current Weapon node and extended it and added a get ammo node and hooked it to the return node. In the float to string conversion node disabled grouping.

**Aiming down sights**

We want the player to aim with his sight along the muzzle when we click the right-mouse button.

Went to the project settings and in the input section made a new binding called AimDownSight and set the input key as the right mouse button. In the third person character blueprint in the viewport added a skeletal mesh and made it the child of the mesh and then set the parent socket as Weapon Attach and also set the skeletal mesh’s mesh as AK but we did so only to get a preview inside the viewport of how the end result would look like and how the camera needs to be positioned. Set the visibility of the mesh as hidden during game so that the weapon we spawn doesn’t intersect with the skeletal mesh. Added a new camera and made it the child of the skeletal mesh. Named the camera as ADSCamera. Set the parent socket of the camera to Muzzle so that the camera is hooked to the muzzle. Adjusted its transform.

In the event graph added an AimDownSight node and added in a reference to the default camera which is the Follow Camera. Extended it and added a deactivate node and hooked its input exec pin to the AimDownSight node’s pressed pin. Dragged in a reference to ADSCamera and extended it and added an activate node and hooked its input exec pin to the deactivate node’s output exec pin. As we don’t want the player to aim while running, added character movement reference and added a set max walk speed node and set its value to 300. Hooked its input exec pin to the activate node’s output exec pin.

Added in a reference to the ADSCamera and extended it and added a deactivate node and hooked its input exec pin to the released pin. Added in a reference of the FollowCamera and extended it and added an activate node and hooked its input exec pin to the deactivate node’s output exec pin. Since if the player isn’t aiming, he can run. So added a character movement reference and added a set max walk speed node and set its value to 600 and hooked its input exec pin to the activate node’s output exec pin.

**Spawning the muzzle flash**

We want a flash effect to appear at the end of the muzzle whenever we fire. So added the Infinity Blade effects pack to the project and opened the AK47 skeletal mesh and right clicked the muzzle socked and in the preview mesh option selected the muzzle flash mesh to see if the placement was right. The scale and rotation is insignificant as they can be handled in real-time. The mesh didn’t need to be set to hidden as it was a preview only mesh.

Opened the Weapon\_Base event graph and extended the set Ammo node’s exec pin and added a cast to third person character node and set the object wildcard to get player character. Extended the exec pin and added a spawn emitter at location node. Extended the as third person character and added a Current Weapon node and extended it and added a get socket location node and set the socket name to muzzle flash and hooked the output to the emitter node’s location pin. Extended the current weapon node and extended it and added a get socket rotation node and set the socket name to muzzle flash and hooked the output to the emitter node’s rotation pin. Extended the return value and added a set world scale 3d node and set the scale values to 0.05,0.05,0.05 respectively.

**Fixing movement states**

Right now, while the player is sprinting and crouch is clicked, he just stop running and starts moving in the crouched position but if he is crouching and then sprinting is clicked he runs while crouching. Not only is this confusing, it’s also unnatural to run at a high speed while crouching. I decided to let the player either sprint or crouch but not both at the same time. To do this all we need to do is to make the crouching variable to false whenever sprint is clicked and sprinting variable to false whenever crouch is clicked.

To do that went to the third person character blueprint and in the crouching code section extended the pressed and added a set isHeSprinting node and set its value to false and hooked it to the rest of the nodes. Went to the sprinting code section extended the pressed and added a set isHeCrouching node and set its value to false and hooked it to the rest of the nodes.

So, whenever crouch is pressed while sprinting, sprinting stops and crouching starts and vice versa.

**Setting up Ammo Reloading**

We want a max count of the ammo we have and reload according to the clip size. But there are some conditions. If the current ammo is equal to the clip size, then we should not be able to reload. If the max ammo is less than the clip size then that count should be the ammo and max ammo should now be zero. If the max ammo is greater than clip size then on every reload, the max ammo should reduce by the clip size.

Opened the Weapon\_Base blueprint and made 2 integer variables – MaxAmmo and ClipSize with default values 90 and 30 respectively as it’s an AK47. Made a new function called reload and extended it and added a branch node. Extended the condition and added an integer==integer node and hooked Ammo and ClipSize to it. Extended the false pin and added a branch node. To the condition added an integer>integer node and to the first hooked MaxAmmo and to the second hooked the ClipSize. Extended the false pin and added a set ammo node and to the value pin added a get MaxAmmo node and extended the exec pin and added a set MaxAmmo node and set its value to 0. Extended the true pin and added a set Ammo node and to its value pin added a get ClipSize node and extended the exec pin and added a set MaxAmmo node and extended its value pin and added an integer-integer node and to the first input added a MaxAmmo node and the second added a ClipSize node.

Opened project settings and in the input section added an event binding called Reload and set the input key as R. in the third person character blueprint added a Reload node and dragged in a current weapon reference and extended it and added a reload function node and hooked the pressed pin to the reload function node.

Opened the HealthHUD and below the ammo text added a new text and set the size and color and anchored it to the top right-hand corner and set the text default value to 0. Added a binding to the text and extended the get text and added a get third person character and set the object wildcard to get player character. Extended the as third person character pin and added a get Current Weapon node and extended it and added a get MaxAmmo node and hooked it to the return value and unchecked the grouping option. Hooked the third person character exec pin to the return node’s input exec pin.

**Conservative Reloading**

In games like Battlefield on reloading only the no: of bullets needed to get a full clip is added to the ammo. Although it isn’t as realistic, it reduces the strain of keeping track of the ammo count. The logic is to check if the clip size is not equal to the current ammo. If not then check whether the max ammo is less than the no: of bullets needed to make a full clip. If not then reduce the max ammo by this amount and then set the ammo to full clip. If yes then add the current ammo with the max ammo and then set max ammo to 0.

Opened project settings and in the input section made a new event binding called ConservativeReload and set the input key to C.

Went to the Weapon\_Base blueprint and made a new function called ConservartiveReload and extended it and added a branch node and set the condition as and integer==integer node and the input pins as Ammo and ClipSize. Extended the false pin and made a new branch node. As we are going to need the difference between the clip size and the ammo frequently, made a new integer variable named Difference. Extended the branch node’s condition pin and added an integer>integer node and extended the first pin and added as set Difference node and extended its value pin and added and integer-integer node and to the first input added a ClipSize node and the second added an Ammo node. To the second pin of the integer>integer node added a get MaxAmmo node. Extended the true pin and added a set Ammo node and extended the value pin and added an integer+integer node and to the first pin added a get ammo node and to the second added a get MaxAmmo node and extended the set Ammo node’s exec pin and added a set MaxAmmo node and set its value to 0.

Extended the false pin and added a set MaxAmmo node and extended its value pin and added an integer-integer node and to the first added a get MaxAmmo node and to the other added a get Difference node. Extended the set MaxAmmo node and added a set Ammo node and extended its value pin and added a get ClipSize node.

The first branch node that checks the clip size == ammo can be avoided as then the difference is 0 and we get the same result. Although that is one condition less during a has ammo situation, having the condition means avoiding calculations during a no ammo situation. You can also optimize it in the very first condition by first finding difference and then checking if that is equal to 0.

Opened the third person character blueprint and added a ConservativeReload node. Dragged in a reference to the Current Weapon. Extended it and added a ConservativeReload function and hooked the pressed pin to the node’s exec pin.

**Reloading animation**

There are 2 ways that we can add a reload animation.

The simple less complex, less fuctionality way:

Open the Walk\_Run\_BP and in the anim graph added a new state called Reload and made arrows back and forth Crouch, Idle and Walk\_Run. It’s hard to reload while sprinting so didn’t make arrows from it to make it look natural. Clicked the Crouch state and to the final animation added a crouching animation. To trigger the animation, we need a variable to decide whether the person is reloading or not. So went to the third person character blueprint and made a Boolean variable isHeReloading and went to the reload code section, extended the pressed and added a set isHeReloading node and set it to true and extended it and added a delay node with delay time 2.4s as that time was perfect to make the animation look natural. Extended the delay node and set the isHeReloading node to false and hooked it to the rest of the code i.e. the reload function. Did the same procedure for conservative load too. In the Walk\_Run\_BP we need a local variable to tell if the player is reloading. So went to the event graph and added a new pin in the sequence node. Extended the Then4 pin and added a cast to third person character and set the object blueprint to try get pawn owner and extended the as third person character and added a get isHeReloading node and extended it and promoted it to a local variable named ReloadingOrNot. Then went to the anim blueprint and set the transition conditions. From one state to reload, extended the can enter transition pin and added a Boolean==Boolean node and to one added a get ReloadingOrNot and set the other as true. Did the same for transition from reload to other states only in this case the other value is set as false.

The problem with this method is that the reloading animation happens whenever you press the reload keys. Even if you have no ammo or max ammo, the animation takes place with no effect on the output. Although this is permissible in many games, to make it interactive and friendly, we use the more complex way for now. But this method has an added advantage that the firing can happen only after reload animation in case of the reload function but still it can happen while reload animation if there are bullets in the ammo if you are using conservative reload.

The complex way with increased functionality:

Follow the initial and final steps of making a variable and setting up the animation blueprint and transitions. Open the Weapon\_Base blueprint and to both the set nodes of the reload function added a cast to third person character node and set the object wildcard as get player character. Extended the as third person character and added a set isHeCrouching node and set it to true. As we need a delay next but can’t add delay inside functions, we extend the exec pin and add a return node and extend the ouput value of isHeCrouching node to the return node. Did this for both the branches.

In the third person character blueprint went to the reload code section. Now we see a output pin on the reload function node. Extended the pin and added a branch node and hooked the input exec pin of the branch node to the output exec pin of the reload function node and extended the true pin and added a delay node and set its value to 2.4 as that is what’s needed to make this particular animation look natural. Extended its exec pin and added a set isHeReloading node to false.

Did the same procedure for the conservative reload part.

**Fixing the firing system**

As of now we are able to fire while sprinting and while reloading. Although it’s natural to be able to do the former, it’s a bother and a waste of bullets as you can’t aim while sprinting. Also, it’s unnatural to fire while reloading. To stop this from happening in the third person character blueprint went to the firing code section and extended the pressed and added a branch statement to the condition added a Boolean OR node and to the 2 values added the get isHeSprinting and a get isHeReloading node. Hooked the false pin of the branch node to the rest of the code. That way even though we try to fire it first checks whether we are sprinting or reloading and fires only if he isn’t doing either of the two tasks.

**Adding a static Crosshair**

Didn’t actually do this part but this is how we add a static crosshair. The only reason I didn’t use this method because I used a dynamic crosshair asset which could expand while moving. Some static crosshairs can be found on Virtus’s site. However, any transparent background image will do. Drag it into the content browser and open the HealthHUD and add an image and anchor it to the centre and transform it accordingly. Set the image as the imported crosshair. With this the crosshair will appear in the centre of the screen during gameplay but this could interfere with situations like sprinting. To solve this problem, add a binding to the image’s visibility and extend the pin and add a cast to third person character node and set the object wildcard as get player character. Extend the as third person character and add a get isHeSprinting node and add a Boolean==Boolean node and set the other value as true. Extend the output value and add a branch node and hook its input exec pin to the third person character’s output exec pin. Extend the true pin and add a return node and set it to hidden and extend the false and hook it to the other return node and set it to appear.

**Adding the dynamic crosshair**

The crosshair used in the game is a dynamic asset which expands on movement. Couldn’t import it directly so cut and pasted it in the directory. Opened it and it had three variables – crosshair\_spread, crosshair\_thickness and crosshair\_lenght. The event graph contains nodes that help set it. This asset is a widget blueprint.

Opened the third person character blueprint and int the event begin play node section we see the code that displays the HealthHUD to viewport. Extended this and added a create widget node and set the class as WBCrosshair and extended its return value and promoted it to a variable called CrossHair and extended it and added an add to viewport node. Hooked its exec pin to the rest of the code. During gameplay the crosshair appeared but it was too small and wasn’t dynamic.

To add dynamicity went to the third person character blueprint and in the even tick section i.e. the armour regeneration section. Initially, the delay node preceded the sequence node but this favours only the regeneration part as only that needs a delay. So made the sequence node precede the delay node and then added in a reference to CrossHair and extended it and added a set cross hair spread node. Added a get velocity node and extended it and added a vector length node and extended it and added a Map Range Clamped node and set the values of in Range A, In Range B, Out Range A and Out Range B as 0, 450, 5, 80 respectively and hooked its return value to the set cross hair spread value.

In the WBCrosshair blueprint set the crosshair length to 30 and thickness to 4. Now during gameplay, the crosshair has a good size and also expands while running.

**Picking up ammo**

The logic for the pickup is that once the player overlaps it the max ammo has to increase, a sound could be played, particle effects may be shown but finally it has to destroy itself i.e. get removed from scene.

To make a pickup we just set up the functionality for now. Made an actor blueprint named Pickup and opened it and added a sphere and set its collision to overlap all and in the event graph went to the event begin overlap node and extended the other actor and added a cast to third person character node and extended the as third person character and added a get Current Weapon node and extended it and added a set MaxAmmo node and hooked its input pin to the third person character output exec pin. To the value pin added an integer+integer node. To one pin added a get AmmoSize node and to the other extended the Current Weapon node and added a get MaxAmmo node and hooked it into the second input. Extended the exec pin of the set MaxAmmo node and added a spawn sound at location node and set the sound as fire sparks and extended the location pin and added a get actor location node. Extended the exec pin and added a delay node and set the value to 1. Extended the return value of the spawn sound at location node and added a stop node and hooked its input exec pin to the delay node’s output exec pin. Extended this and added a destroy actor node.

The destroy actor node should be placed before the sound nodes in my opinion if the delay time is too long. If the audio takes a long time to play, the player could overlap more than once which is unnatural and not what we want.

**Making Targets**

Set up the basic functionality for a target system in this part. Opened the bullet blueprint and set the bullet collision to overlap all and made a new float variable Damage and set the default value to 0.1. Made a new blueprint character class and in it set the skeletal mesh as the unreal mannequin for now. Adjusted the transform such that it just fit the capsule else the effects would be awkward and also set the rotation such that the face is in the direction of the arrow. Set the collision to overlap all. Opened the event graph and made a float variable Health and set the default value as 1. Extended the on event begin overlap and added the cast to bullet node and extended it and added a set Health node and extended the value pin and added a float-float node and to the top pin added a get Health node and extended the as Bullet node and added a get Damage node and hooked its value to the second pin. Extended the output exec pin of the set Health node and added a Branch node and extended the condition pin and added a float>=float pin and to the first value hooked the return value of the set node and set the second value to 0. Extended the false pin and added a destroy actor node.

What this does is to reduce the health by the damage value and if it ever reaches 0 or below, destroys the actor.

**Making the enemy follow the player using AI**

For now, what’s needed is for the enemy to run towards the player whenever he sees him. To use AI first the computer needs to know where all it can and can’t move and for that we use a navigation volume and we insert it into the scene and press P to see the path that can be travelled and it’s shown in green and the red indicates collision i.e. the regions where there are obstacles. Transformed the navigation box volume such that the green covered the entire ground floor.

Opened the Zombie blueprint and in the viewport added a PawnSensing component and adjusted the vision radius and peripheral vision. Then clicked the on see pawn button and went to the event graph. Extended the pawn pin and added the cast to third person character node and extended the exec pin and added an AI move to node. Set the pawn pin to a self node and there are 2 options from here. If you want the zombie to move to a particular location on seeing you set the destination pin. If you want it to move towards you in real-time, extend the as third person character pin and hook it to target actor. I did the latter in this project.

**Fixing aiming glitches**

When the player tried aiming down sight there were a few glitches. The crosshair didn’t disappear. When the player tried aiming during reloading the camera shifted to the ADCamera while still reloading which was unnatural. While the player tried to sprint while aiming the camera shifted to ADCamera which was pointed to the ground so that too was very unnatural.

Opened the third person character blueprint and made Boolean variable isHeAiming. To make the crosshair disappear opened the WBCrosshair and the way it works is using 4 parts the bottom, left, right and top. Selected the bottom and added a binding to its visibility. Extended the pin and added a cast to third person character node and set the object wildcard to get player character. Extended the as third person character and added a get isHeAiming node. Extended the third person character node’s exec pin and added a branch node and extended the true pin and added a return node and set it to hidden and extended the false pin’s return node and set it to visible.

Set the bindings of the other 3 parts to the function which was created in the above section.

In the third person blueprint’s aim down sight section, extended the pressed key and added a set isHeAiming node and set its value to true and hooked it to the rest of the code. Extended the released pin and added a set isHeAiming node and set its value to false and hooked it to the rest of the code.

To fix the reloading issue extended the pressed pin of aim down sight and added a branch node and to the condition added a get isHeReloading node and extended the false and added a branch node and to the condition hooked a get isHeSprinting node and extended the true and added a set isHeSprinting node and set its value to false and hooked it to the rest of the code. Extended the false and directly hook it to the rest of the code.

This solves only part the issue as this much leaves the player unable to aim while sprinting or reloading but he can still sprint and reload while aiming. In this case just setting the isHeAiming to false in their respective pressed pins wont solve the problem as the camera goes to ADCamera while the mouse is clicked and comes back only when released. So we need to give the entire released functionality to pressed pins of sprinting and reloading so that the camera goes back to the follow camera. To make the code compact, selected all nodes from the released section of the aim down sight section and collapsed them to a function named NotAimingFunctionality. Extended the pressed pins of both sprinting and reloading and added the NotAimingFunctionality node and hooked its exec pin to the rest of the node.

**Smarter AI actions**

To make the zombie chase you whenever you shoot them. In the Zombie blueprint go to the begin overlap part. For now, the branch node only checks the health and if it’s less than zero destroys the actor. We want the zombie to run towards the actor if it’s shot and is still alive. We also want to add a particle effect whenever the zombie is struck. Extended the false pin and added a spawn emitter at location node and set the emitter to P\_Impact\_Shield\_Fire which is in the Infinity Blade effects pack. Went back in the graph and found a reference to the bullet and extended it and added a get actor location node and hooked its value to the emitter node location. Extended its exec pin and added a delay node as it’s unnatural to know immediately the attacker when you have been shot. Extended this and added a cast to third person character and set the object wildcard to get player character as we want the zombie to run to player location. Extended the exec pin and added an AI move to node and extended the pawn and added a self node. Extended the as third person character and hooked it to Target Actor.

Whenever shot, the zombie after a brief delay runs towards the actor. Also, a particle effect appears where the bullet strikes the zombie. If you want you can make the zombie rotate to look around and if he sees the attacker run towards him or if he doesn’t make him hide etc.

**Random Roaming**

We want the zombies to roam around in a random way whenever it doesn’t see the player. To do this we use a navigable radius which decides where all the zombie can move. The collisions are calculated automatically so the zombie doesn’t collide with other objects.

Opened the zombie blueprint and made a new Boolean variable CanHeSeeHim. Went to the on see pawn section and between the cast to third person character and AI move to added a set CanSeeHim node and set the value to true. Extended the success pin and added a set CanSeeHim node and set its value to false.

To tell the AI to roam around, made an event tick node and added a delay node and set the delay value. Added an AI move to node and extended the pawn and made a self node. Extended the location and made a Get Random reachable point in radius node which makes it possible to move only wherever there can be no collisions. Set the radius to 250 as that seemed to be a good value. To get the origin there are 2 possible ways – get the location of the zombie or the location of any other entity such as the player. In the game we use the zombie’s own location to be the origin. For that added a get actor location which is already set to self and hooked it to the Origin of the Get Random Reachable Node.

**Smoothening Gun Movement**

There are some glitches in the gun movement. When the player moves right or left, the gun moves slightly in that direction. This isn’t good as the gun should be steady. A suggested trick is given below but that didn’t work for me so I haven’t used it. This is the layered animation method.

The trick is to make the upper body idle while running or walking. Opened the Walk\_Run\_BP and in the anim graph open the Walk\_Run state and extend the Walk\_Run\_BS node and add a layered blend per bone node. To the base pose hooked the Walk\_Run\_BS and extend the other pose and add the idle\_aiming animation. Set the blend weight and blend depth accordingly. Add a layer and set the bone to the bone you want to make idle Spine2 recommended.

This didn’t work for me. Must learn more about layered animation.

**Setting up an Objective System**

Set up a simple objective system that displays the objective on top of screen and changes as soon as player overlaps with the PlayerObjective actor. Opened the third person character blueprint and made a String variable HisObjective. Set the default text as “Scout Ground”. Made a new actor blueprint and added a box collision mesh. Opened the event graph and extended the on event begin overlap node’s other actor pin and added a cast to third person character node. Extended the as third person character and added a set HisObjective node and set the value to “Kill Zombies”. Opened the HealthHUD and added 2 text blocks and set text of 1 to Objective and to the other’s text created a binding and extended it and added a cast to third person character node and extended the as third person character and added a get HisObjective node and hooked it to the return value.

Dragged in the actor to the scene and during gameplay, on overlapping in the objective changes from “Scout Ground” to “Kill Zombies”.

**Modifying the Score HUD**

Dragged in the HUD images of virtus’s asset pack and added a few images to HealthHUD to get a border, health and armour icons and grenade icons. The main change made was to set both the health and armour bar’s color to white and added a background image – no\_fill and fill image – fill so whenever there’s a change in the health or armour the fill image gets filled over the background image progressively.

**Killfeed**

Made a kill feed that pops up every time a player is killed. Made a new UI widget named KillFeed and opened it and inserted a border and filled it with black color. To get its position right copied it and placed it in the HealthHUD just below the border and copied the co-ordinates and pasted it in the KillFeed border’s coordinates. Inserted 2 texts one saying “You Killed:” and the other saying “player01” but that is to be changed according to the player. Gave names to all three components.

Made a new animation named Killfeed. Inserted a track for the border and at 1 sec changed its render opacity to 0 and transform’s y coordinates to 115 so that it moves down while fading. Did the same for the other 2 text components. Opened the event graph and extended event construct and added a play animation node. Added a reference to Killfeed and hooked it to the animation object.

Opened the Zombie class and in the event graph went to the part where the zombie is killed i.e. destroy actor node and extended it and added a create widget node and extended it and added an add to viewport node.

Mini map

Open third person player blueprint. In the viewport add a camera and take it straight up and then rotate it so as to point directly down. Add a screen capture 2D component and make sure it’s a child of the added camera. Named the camera MapCam.

In the content browser made a Render Surface component under materials and textures section. Selected it and in the third person character blueprint clicked the arrow button in the screen capture 2D’s screen capture section’s Texture Target section. The arrow key means import object selected in content browser. In the content browser made a new material named Minimap\_Material and dragged in the texture. In the left window, changed the sample source option to From texture asset so that changes the output node to a node with only the necessary pins. Hooked the RGB color pin of the texture to the base color pin although modifications could be done to change the look.

Opened the HealthHUD and added 2 image components. Placed one at the top-left corner and changed its size. With the material selected in the content browser pressed the arrow key in the image option under the appearance subsection and that gives the mini map.

However, I wanted an arrow to point in the direction where the player is headed and to replace the player with the arrow icon. The least complex way to do this is by making use of the fact that the MapCam always points downwards at the player so at all times, the camera will have the player at its centre so just adjust the Z order of the image so that it’s on top of the mini map image and then place the arrow icon image at the centre directly on top of the player so now the arrow moves and turns whenever the player does.

One major glitch with this procedure is that now during run-time whenever the left-mouse button is clicked, the MapCam is activated instead of the ADSCamera. Solved this by going to the third person camera, selecting the MapCam and unchecking the auto-activate option under the activation section.

**Match Timer**

The logic is to have 2 variables. The second should decrement from 59 until it reaches 0 and should then be set to 59 if minute>0. The minute should get decremented every time second reaches 0. When minute becomes negative the game should stop.

To make a match timer opened the HealthHUD and added an image and set it to Virtus’ timer texture. Added 3 texts for min, colon and sec. Set their default values to 00, :, 0 respectively. Opened the third person game mode and in the event graph made 2 integer variables Minutes and Seconds and set their default values to 15, 00 respectively. Added an event tick node and extended it and added a delay node and set delay to 1s. Extended it and added a set Seconds node and in the input added an integer-integer node and to its 1st input added a get Seconds node and set the second to 1. Extended the output pin of the set Seconds node and added a branch node and to the condition added an integer<integer node and to the first input added a get seconds node and set the second value as 0. Extended the true pin and added a set Minutes node and extended the input pin and added an integer-integer node and to the first input added a get Minutes node and set the second to 1. Extended the output of the set Minutes node and added a branch node and to the condition added an integer-integer node and to its 1st input added a Get Minutes node and set the second as 0. Extended the true pin and added a Game Pause node.

**Hit Marker**

We want the player to know when he hits the target. Most games have some form of animation like text, sound, hit markers etc. Here we use a hit marker that appears and fades away whenever a bullet hits the target.

Made a new widget HitMarker. Added an image and set it to Virtus’ hit marker texture. Set the image’s render opacity to 0. Made a new animation and named it HitMarker and added a track and selected the image. Added a new track and selected render opacity. At 0.25s, opacity = 1 and at 0.50s, opacity = 0. Opened the event graph and extended the event construct node and added a play animation node and dragged in a reference of HitMarker (the animation) and hooked it to the animation. Opened the Zombie class and went to the region where the bullet hits it i.e. when its health is decremented at the beginning of the workflow added a create widget node and set the class to HitMarker and extended the return value and added an add to viewport node. The hitmarker fades in and out whenever the zombie is hit.

**Starting Weapon Switching (HUD)**

Whenever 1 is pressed on the HUD “AK-47” and its icon must be shown and when 2 is pressed the 2 should be replaced by “M4A1” and its icon.

Opened the project settings and in the input section added 2 event bindings PrimaryWeapon and SecondaryWeapon which are activated by num 1, 1 and num 2, 2 respectively. Opened the third person character and made an integer variable called WeaponSelected and set its default value to 1. Made a PrimaryWeapon node, extended it and set WeaponSelected to 1. Made a SecondaryWeapon node, extended it and set WeaponSelected to 2.

Opened HealthHUD added a text and set its default value to AK47. Created a binding to its text and extended the node and added a cast to third person character and set object wildcard to get player character and extended the as third person character and added a get WeaponSelected node. Extended the output pin of the cast to third person character node and inserted a Switch on int node and added pins by clicking the plus sign. Added pins 1 and 2. Extended the pin 1 and hooked it to a return node and set the return value to AK-47. Extended the pin 2 and added a return node and set its return value to M4A1.

Added 2 images and set it to Virtus’ AK-47 and M4A1 textures respectively. Placed the images such that they overlapped each other. Added a binding to the visibility of the AK-47 image and extended the pin and added a cast to third person character pin and extended the as third person character and added a get WeaponSelected node. Extended the output pin of the third person character node and added a Switch on int node. Added pins 1 and 2. Extended 1 and hooked it to the return node and set the return value to visible and extended 2 and added a return node and set the return value to hidden. Did the same procedure for the M4A1 image only difference here is that instead of 1 pin of AK-47, 2’s return node value is visible and 1’s return node value is hidden.

**Weapon Pickup**

We want the AK-47 to be the primary weapon and the M4A1 to be available when the player walks into a pickup. So before getting a pickup pressing 2 would be useless.

Imported Virtus’ M4A1 asset and diffuse texture and made sure to import the skeletal mesh. To make the material for the gun opened its material and dragged in the texture. Hooked the RGB to the base color and the emissive color (for shininess at the right places). Extended the metallic and added a constant node and set its value to 1. Extended the roughness and added a constant node and set its value to 0.2.

Made a new blueprint actor called M4A1\_Pickup and added the M4A1 skeletal mesh. Opened the third person character blueprint and made a Boolean variable called HasWeaponBeenPickedUp. In the M4A1\_Pickup after setting the gun’s collision type to Overlap all, went to the event graph and extended the on object begin overlap and added a cast to third person character node and extended it and added a set HasWeaponBeenPickedUp node and set it to true. Then extended its output pin and added a destroy actor node.

Opened the third person character and from the SecondaryWeapon input event section, between the pressed and set WeaponSelected nodes added a branch node and set its condition to get HasWeaponBeenPickedUp and extended the true pin and hooked it to the rest of the code to ensure that the gun can be used only if it has been picked up.

**Weapon pickup message**

Made a new widget called M4A1\_Pickup and opened it. Added a text saying “Picked Up M4A1” and set its render opacity to 0. Added an animation named Popup and added a new track and clicked the text. Added a new track and selected render opacity and set the values at 0.5s, .1s and 1.5s and 2s as 0, 1, 1 and 0 respectively. Went to the event graph and extended the event construct and added a play animation node and dragged in a reference to Popup and hooked it to animation. Opened the M4A1\_Pickup blueprint and went to the part just before the destroy actor is called and in between it added a create widget node and extended the return value and added an add to viewport node.

**M4A1 weapon setup**

As we created the AK47’s Weapon\_base as a base class for all future guns, it’s simple to add new weapons. All we need to do is copy the Weapon\_base of AK47 folder and paste it in the M4A1 folder and rename it to Weapon\_M4A1. Opened the blueprint and in the viewport added a new skeletal mesh and named it M4A1. Didn’t remove the AK mesh and it was used as a reference to position and scale the new gun. The guns were placed such that their muzzles coincide so the aim and other stuff don’t change. To get the bullets to come out of the muzzle, opened the M4A1 skeletal mesh and went to its skeleton and added a new socket named Muzzle. Adjusted its position and scale using the muzzle flash asset which was added using the preview. Special care is to be taken so as to give the same socket names to both guns.

To spawn the gun into the scene opened the third person character and went to the event begin play node and went to the part where the AK47 is spawned. Copied the whole section and hooked it to the end of the code. However, to spawn M4A1 instead of AK47, located the part where the value was promoted to a local variable named Current Weapon. This was for AK47 so deleted the Current Weapon reference and extended the value and promoted it to a new local variable called SecondaryWeapon and hooked it to the rest of the code. During gameplay, both guns were spawned at the same location.

**Weapon Visibility**

To make only one gun visible at a time, opened the third person character and located the section where M4A1 is spawned and added in a reference to the SecondaryWeapon and extended the reference and added a visibility node and set the visibility to null and then hooked it to the rest of the code. Now by default only AK47 is shown.

Went to the PrimaryWeapon input node and at the end of it added in a reference to the SecondaryWeapon and extended the reference and added a visibility node and set the visibility to null and then hooked it with the rest of the code. Added in a reference to CurrentWeapon and extended it and added a visibility node and set visibility to true and hooked it to the rest of the code. Went to the SecondaryWeapon input node and at the end of it added in a reference to the CurrentWeapon and extended the reference and added a visibility node and set it to false and then hooked it to the rest of the code. Added in a reference to the SecondaryWeapon and extended the reference and added a visibility node and set it to true and then hooked it to the rest of the code.

Now whenever 1 is pressed AK47 is visible and when 2 is pressed after a pickup M4A1 is visible. Remember to add in the reference first, extend in and then add the visibility node so that the intermediate component is the name of the skeletal mesh and not anything else.

**Displaying secondary ammo**

To display secondary ammo details when secondary weapon is being used. Opened the HealthHUD and in the text box that displays ammo went to the binding function and extended the output pin of cast to the third person character node’s output pin and added a switch on int node and added pins 1 and 2. Extended the as third person character pin and added a get WeaponSelected node and hooked it to the condition pin. Extended 1 and hooked it to the rest of the code as that was to display the AK47 details. Extended the as third person character and added a get SecondaryWeapon node and extended it and added a get Ammo node and extended it and added it to a return node and hooked the input pin of this 2nd return node to the 2nd pin of the switch node.

Did the same procedure for the max ammo text box’s binding. However, there is a new glitch. Although the respective guns ammo is displayed, both the guns used the ammo of AK47. That’s because the firing functionality was made with AK47 (CurrentWeapon) references. This needed to be changed.

**Resetting the firing and reloading for weapon systems**

As we are using 2 weapons now, at the beginning of the fire, the reload and the conservative reload functionalities we need to use a switch on int to check which weapon has been selected and then if it’s 1 the functions must act on AK47 and if it’s 2 they must act on the M4A1. Just implement the switch functionality and use the same code for pins 1 and 2 only in 2 the object references must be changed to that of the M4A1. After that go to the Weapon\_M4A1 footprint and change all parameters of AK47 to parameters of M4A1.

With the functionality set up the it’s seen that the HUD displays ammo and max ammo of the AK47 at all times that’s because the text binding always checks the AK47. Add the same switch functionality. For 2 make sure that only parameters of M4A1 are used.

Did this and the screen displayed the info of the current weapon and on firing the bullets were spawned and reduced from the respective weapon’s max ammo count.

**Weapon line tracing**

The firing mechanism we initially used is the projectile system. This isn’t 100% accurate and takes a long time to adjust. We need something that is 100% accurate and looks natural. Although projectile system could be less complex for increased realism, we use line tracing i.e. the bullet travels in a straight line from wherever we want it to.

For learning purposes, I have used the line trace method to the AK47 and the projectile method for the M4A1. Went to the third person blueprint and went to the firing section and double clicked the fire function of the AK47 gun. Removed the link from the Fire custom event node and moved it up. Extended it and added a cast to third person character node. Extended the output pin of the node and added a line trace by channel node. We need the bullets to spawn from the centre of the screen i.e. Follow Camera’s position and move some distance forward and then stop. So extended the as third person character and added a get follow camera node and extended it and added a get world transform node (as we are going to need location and rotation but I feel complexity could be reduced if they are inserted separately) and split the struct pin. Extended the location pin and hooked it to the start pin. Extended the rotation pin and added a get forward vector node and extended it and added a vector\*float node and set the float to a value say 1500. If there are multiple weapons with line trace, it would be easier to make a local variable and store the value in it. So made a float variable MaxDistanceLineTraceGoes and set the default value to 1500 and hooked it to the second pin of the vector\*float node. Extended the node’s output pin and added a vector + vector node and hooked the location pin to the second value. Hooked the output of the node to the end pin of the Line trace by channel node. Went to the projectile bullet code and copied all the code after spawn bullet node and pasted it after the line trace by channel node and hooked it to the output pin of the channel node.

For debugging purposes, set the line channel to visible and the draw debug type to persistent. On shooting lines starting from the centre of the screen ended 1500 units in the direction of the cross-hair (as the cross hair is in the centre of the screen). However, the zombies didn’t die on being hit. To test this, extended the output pin of the line trace by channel node and added a print string node and extended the out hit pin and added a break hit result node and hooked the hit actor pin to the in string pin of the print string node. Now during playing it’s seen that the zombies are not being hit so set the zombie mesh from overlap all to block all and now whenever a collision occurs a message is shown. However, at this stage the zombies can only be killed by an M4A1 as the kill function depends on the overlap of an object.

**Ray tracing with aiming down sights**

One glitch with the above method is that the bullets come according to the position of the follow camera but when the player is aiming the camera shifts to the ADSCamera so the bullets don’t come from the new centre but still comes from the follow camera. So, we need to change the camera in the fire function when the player is aiming while firing.

Opened the fire functionality of the AK47 and in between the cast to third person character and line trace by channel added a branch node. Extended the as third person character and added a get is he aiming node and hooked it to the condition of the branch node. Hooked the false pin to the line trace by channel node. Copied all the nodes from the branch node to the print string node and pasted it above and near the true pin. Extended the as third person character and added a get ADSCamera node and hooked it to the get world transform node. Hooked the true pin to the newly added line trace by channel node. Hooked the output pin of the newly added print string node to the set node.

**Damaging zombies with line trace**

Although the zombie collision was changed to block all, they still didn’t get damaged even when the hit component function was used. This is because the line tracing function has nothing to do with bullets. It just traces a line vector. But from that we can get info if the vector collides with other stuff like actors, components, time etc. so we need to make use of this info to make a new functionality for damaging the zombie.

The mechanism to do this is almost same like the previous time. Went to the zombie class and copied all the shooting functionality except the event begin overlap node and pasted it in the Weapon\_base blueprint. Deleted the print string node as it was for debugging. As we need bullets to be reduced from max ammo irrespective of whether they hit the zombie or not, the output exec pin of the line trace by channel node is hooked to the set ammo node. But now we need a copy of the bullet reduction and emitter spawning functionality for the 2nd line trace by channel node too as the actor given by the two of them will be different as both use different cameras. After the set world scale 3D node which is the last node of the bullet reduction and emitter spawning functionality, hooked its output exec pin to the 1st copy of the cast to bullet node which is the first node in the zombie damage functionality. Replaced the cast to bullet node with the cast to zombie node and as the object hooked it to the hit actor pin of the break hit result node. The zombie class’ local variables couldn’t be used directly so give them reference to the zombie object by using cast to zombie nodes or by hooking them to the hit actor pin of the break hit result node. Did the same for the 2nd line trace.

**Crouching adjustments**

One major glitch is that right now we only have a crouching animation and that is not going to change the collision nature. When we crouch the collision height should decrease. Unreal has an inbuilt crouch and uncrouch function. To access it went to the third person blueprint and went to the viewport. Selected the character mesh and in the right-hand window under the Nav Movement, clicked the Movement capabilities so the list is extended and made sure the Can Crouch option was checked. It wasn’t so I checked it.

Then went to the event graph to the crouching section and after the set max walk speed node of the pressed section added a crouch node and after the set max walk speed node of the released section added an uncrouch node.

As the sprint section unchecks the crouch before executing the sprint functionality and since the crouch function that was activated during crouching isn’t deactivated if we directly try to sprint, this may cause problems so after the set is he crouching node in the sprint functionality added an uncrouch node. Now the character is able to go under things which he couldn’t while standing.

**Setting up zombie animations**

Imported the zombie mesh provided by Virtus and after that imported the 4 zombie animation files and, in the popup, made sure that the mesh was set to the zombie skeletal mesh. Made an animation blend space named ZombieBS and named the horizontal and vertical axes as direction and velocity and set their max. limits to 100, 600 respectively. Dragged in the idle animation and placed it at 0,0 the walking animation at 0,150 and the running animation at 0,300. The interpolation time of both axes was set to 1 to get a smooth transition effect. Made a new animation blueprint named ZombieAnimBP and opened it and inserted a machine state node and hooked it to the output node. Double clicked on the machine state and went to the anim graph and extended the entry node and added a new state and named it moving. Opened the state and dragged in the ZombieBS and hooked it to the output node. To get the location and speed, went to the event graph and from the event blueprint update animation node added a cast to zombie node and extended the object wildcard and added a try get pawn owner node. Extended the as zombie node and added a get actor rotation node and extended the zombie node and added a get velocity node. Extended the get actor rotation node and added a calculate direction node and hooked the output of the velocity node to the input pin of calculate direction’s velocity node. Extended the output pin of the calculate direction node and promoted it to a local variable and named it direction. Extended the get velocity node and added a vector length node and extended it and promoted it to a local variable and named it speed. Went to the anim blueprint of moving and dragged in a reference of direction and hooked it into the direction pin of the ZombieBS and dragged in a reference of speed and hooked it into the speed pin of the ZombieBS.

Opened the Zombie blueprint and in the viewport set the mesh to the zombie mesh and change the animation class to ZombieAnimBP. Changed the mesh’s size and location to make it fit the capsule and touch the ground.

**Zombie ragdoll death**

To get ragdoll death we need a physics asset. For some reason the imported mesh’s physics asset that is automatically created might have some issues so right-clicked the zombie mesh and clicked create and clicked create physics asset. Opened it and checked if it’s fine.

I have used the ragdoll death just for study purposes so it will get activated only if you kill with the AK47 while aiming down sight. So, went to the zombie death portion of the AK47’s aim down sight part and deleted the destroy actor node and in place of it added a simulate physics node. By default, the target is AK47 so we delete that target and extend a reference of the zombie i.e. from the past cast to zombie node we drag from the as zombie pin and add a get mesh node and hook its output to the actor pin of the zombie node.

So now whenever the zombie is killed with the AK47 while aiming down sight it falls to the ground and can be kicked around but the moment it is shot at with the AK47 without aiming, the zombie disappears as the health is already 0 so the destroy actor node of the AK47 firing without aiming is used.

**Grenade spawning**

Opened project settings and in the input section added a new binding named GrenadeSpawning and set the input key to G. Made a new blueprint actor called Grenade and in it added a static mesh and named it Grenade and set the mesh to a sphere and set the material to M\_Tech\_Panel as it had a slight metallic look. Checked the simulate physics option and set the collision to block all.

Opened the third person character blueprint and in it added a GrenadeSpawning input action node and extended it and added a spawn actor from class node and set the class to Grenade and extended the spawn transform pin and added a make transform node. Extended the rotation pin and added a get actor rotation node. Extended the location pin and added a vector + vector node. Extended the first pin and added a get actor location node and extended the second pin and added a rotation vector node. Extended the B pin and added a get control rotation node and then made a vector variable called RotateOffset and hooked it to the A pin. Set the value of RotateOffset to 300,0,0. At this point the grenade’s spawned 300 units in front of the controller and drops to the ground. To make it natural it needs to fly away and explode.

Opened the grenade blueprint and in the event graph extended the event beginplay node and added an add impulse node. Dragged in a reference to the static mesh (named Grenade) and hooked it to the Target pin. Extended the impulse pin and added a vector \* vector node. Added a get player controller node and extended it and added a get control rotation node and extended it and added a get rotation X vector node and hooked its output to the first pin of the vector\*vector node. Made a new vector variable named GrenadeImpulseMultiplier and set it to 600,600,600 and hooked it to the second pin of the vector\*vector node. Extended the output pin of the add impulse node and added a delay node and set the delay to 2s. Extended its output and added a spawn emitter attached node and dragged in a reference to the Grenade and hooked it to the attach to component pin. Set the emitter template to P\_Explosion. Extended the output pin and added a spawn sound at location node and dragged in a reference to the Grenade and extended it and added a GetWorldLocation node and hooked the return value to the location pin.

**Grenade damage**

To make the grenade damage the actors in it within a certain radius, opened the Grenade viewport and added a Sphere collision component and set its scale according to the range we need and made sure that the sphere collision was a child of the grenade mesh so that it follows the mesh everywhere. Extended the output pin of the spawn sound at location node and added a destroy actor node. Added a get overlapping actors (sphere) node and set the filter to Zombie so that only zombies get affected by the blast and hooked its pin to the target pin of the destroy actor node. Extended the exec pin of the destroy actor node and added a delay node and added a destroy actor node whose target was self so as to destroy the grenade.

**Decorating the level**

Created a new level and deleted the default plane. Added a landscape and made sure the third person character spawn point was above the landscape. With the sculpt tools made mountains and with the erosion tools made a lake in the middle. Made a new material called LandscapeMat and dragged in a grass texture and a gravel texture and added a blend node and hooked the both into the blend node and hooked the result into output color. Also hooked a const node into specular so as to control the level of reflection on the surface. Set the landscape material to LandscapeMat and created a layer info for both grass and gravel and set the blend weights to 0.5,0.5 so that a blend of the both would be applied on the whole surface automatically and then by clicking on individual textures, they could be painted separately. Added a plane on top of the eroded area so as to cover the whole hole and set its material to Lake water and added a post processing vol. so as to cover the entire area beneath it, enabled its colour and set a blue tint to it.

Using foliage added trees and other stuff and placed the zombies randomly in the landscape.