GameDev Diaries

Making an endless runner game

RunBabyRun

Make a 3rd person project RunBabyRun.

**Adjusting camera**

To adjust camera position so as to see the scene ahead of the player, opened the character blueprint map and adjusted camera position slightly above player. The camera and the player together make the third person so no unnecessary pain in making the camera follow the guy.

**Cancelling unnecessary inputs**

The character could change direction with the mouse but that’s not needed for a mobile game so took the graph editor in the character blueprint and deleted the whole mouse input section which enabled user to change direction.

Note – Although cancelling unnecessary inputs is necessary, I’ve found out that it’s much better to do so at the very and as you have lots of freedom to explore the environment and find the potential glitches. The same applies to making the user run forward endlessly that can be done at the end as it makes debugging a lot easier.

**To make the user run forward endlessly**

To make the user run forward endlessly deleted the forward movement not and, in its place, inserted an event tick node and hooked it to the add movement input node. Set the scale value of the add movement input node to a suitable value. Previously, the w button was used to make the character move forward but now it moves with the tick of each frame no matter what.

**To make the loadable tile**

To make reduce lag and increase efficiency we make loadable tiles that can be deleted from memory the moment the actor runs across it and this will make the game more efficient also allowing us to make infinite running areas without affecting memory.

To make a basic running tile with 3 sections. Create your tile material first. Then create a new class. Add in the static mesh in this case a cube. Set the dimensions and assign it the material. If your material had 3 sections giving a running track-like feel then you can leave it at that or else just add in more cubes so that you can tweak them and divide the original cube or use the cube as one tack and the other 2 as separate tracks or whatever.

Then add an arrow component and place it in whatever direction and location you want the next object to load. You can tweak the location and direction whenever you want as you’re making a class.

Here I ran into a bit of a problem it seemed that the physics button of the cube I created couldn’t be checked. So, after a lot of trial I had to delete the entire project and redo it this time with a plane. But it seemed that the plane too didn’t let the character rest on it but it had the physics button which I could enable and I set the collisions on too. But still the moment the player jumped onto the plane he seemed to go up at a high speed. Solved this problem by increasing the mass of the plane. Later after the code to spawn the cubes were completed and the environment was rid of all other objects except the player class. The moment the planes were created they fell down. Solved this by disabling gravity. Now they didn’t fall but if they were placed close to each other they seemed to topple under the weight. Tried to distance them far apart. Now they would topple only when the player comes on top of the plane. Tried to increase weight, disabling radial forces but of no effect.

The divisions were rectangular planes with no physics or collision enabled but when the player stepped on it, it didn’t topple which was very weird because I think the player fell through the cube because physics could not be enabled for it. Very weird, next time should try varying wt. of the cube to solve the problem.

Anyways, disabled the simulate physics option and now it works. No matter how long you stand on it there is no toppling.

**Programming the loading of tiles**

Opened the master tile class and selected the box collision. Scrolled down its properties and clicked the on Component Begin Overlap option. In the event graph the Begin overlap node appears.

So, this is the logic:

When the box collision is overlapped by an actor a fuction which is capable of loading tiles needs to be called. So, from the begin overlap node’s other actor make the cast to third person character node as that is the actor that can trigger the rest of the actions when he enters the box. We make the make tile function in the ThirdPersonGameMode class so in order to access it from the third person character node insert a cast to ThirdPersonGameMode node. Make sure to extend its object and make a Get Game Mode node as the current game mode is the object we need to work on and we need to call the make tile function of that object.

As the make tile function hadn’t been created, went to the ThirdPersonGameMode class and made a new function called make tile. The function graph appeared. So when the function is called we need it to spawn the master tile class in the location of the arrow of the previous class. So we extend the function output pin and make a spawn actor node and set the actor as the master tile. The spawn location is the arrow of the previous tile. Its value varies so we extend it and select create a local variable and name it SpawnLocation.

Now all we need to do is to give the SpawnLocation a value. The value is the location of the arrow of the master tile class so first we need to access the master tile class which we do by extending the return value of the spawn node and create a cast to master tile node. Now we need the arrow in the master tile class and we had name the arrow ‘SpawnPoint’ so we extend the as Master Tile option and create a get SpawnPoint node. We need the location of the arrow so we extend it and make a get World Location node which gives the co-ordinates of the arrow. Extend the main output pin of the cast to master tile node and create a set SpawnLocation node. The SpawnLocation variable is a struct and has the capacity to store location, rotation and scale. All we need it to store is the location. So, go to the value pin of the node hold down alt + right click and select the split struct pin option and the location, rotation and scale gets separated. Now plug in the get World location node’s output to the location input of the SpawnLocation node.

Now that we have the make tile function ready, go to the MasterTile class and in the event graph extend the as third person game mode pin and insert the make tile function.

Now when we begin the game only having one tile before us can be ugly. Instead of that it would be good if at the start of the game a specific number of tiles (let’s say 10 tiles) are created so as to fool the user into thinking it’s endless. So, we go to the event graph of the ThirdPersonGameMode add an event begin play node, extend it and make a for loop node and extend the loop body and make the make tile node. Finally, we leave the first index as 0 and the last index as 9.

**Programming the unloading of tiles**

We are standing on a tile as soon as the box collision begins a new tile is loaded and now, we have to destroy the tile we were standing on after a time delay enough to let the actor move on to the next tile.

We already coded the loading stuff so now to extend the make tile node (as that was the last node) in the Master Tile class and add a delay node, give the delay value. Extend it and make a destroy actor node. Since we are in the Master Tile class which we want to destroy and since the default value of the destroy actor node is self, this is enough. So as soon as the class creates a tile, it waits for a specific time and then destroys itself.

**Switching between lanes**

Went to the master tile class and in the viewport added 3 arrows. To the left-most arrow set co-ordinates as (0, -330, 0) and named it as Lane0. The middle arrow set co-ordinates as (0, 0, 0) and named it as Lane1. The right-most arrow co-ordinates were set as (0, 330, 0) and named as Lane2.

Although for an Android endless runner game, player movement is controlled by swipes, this section is to do the basic setup in the master tile class and to debug the game using the keyboard as the input device.

So, the goal is to shift the player between lanes. This can be easily done by changing location of the player based on his current location. We give the location values to the Lanes 0,1,2 and we can see that if we want to move left just subtract 1 and if we want to move right just add one which will give us the variable which in turn holds the actual location.

Since the player movements are controlled in the third person character class, went to the class and made 3 variables – Lane which is an integer and compiled it and set default value as 1 as the player is initially in Lane1, NewLane which is an integer and the default value doesn’t matter as all further values are controlled by the player input, LaneY which is of float-type but clicked on the tic-tac like option and made it an array. Added 3 elements in the array and set the values as 0 - -330, 1 – 0, 2 – 330. Created a left node that comes under the keyboard section. Similarly created a right node.

When you press left the NewLane has to be given the value of the lane the player is moving into. To find that lane number just subtract the present lane number by one but the minimum value until which subtraction is possible is 0.

From the left node extended and made a set NewLane node and from the New Lane input pin make a clamp node and set maximum as 2 (just a formality) and minimum as 0. From the value pin extend and create and Integer + Integer node and to one variable box enter -1 as that’s the value to be subtracted and from the other variable box extend and add a set Lane node (just drag in a reference of the Lane variable from the left window).

Do the same for the right node.

This much tells the engine to what lane the player has to go to next. If we plug this in to the set world location node of the capsule (the player), the lanes will get switched but as we are directly setting the value the location changes instantaneously and that will look like teleportation. Instead, we need the player to go from one lane to another to make it look more interesting.

For this all we need to do is to make the player gradually move from his initial value to his new value. To set how fast this transition is, we use a timeline and to gradually change values based on the time line we use a lerp.

Inserted a timeline node and named it lerp timeline. Opened the editor by clicking it and added a float timeline by pressing the button on the top. A float timeline is a set of float values which can be selected and adjusted to go from the initial value to the final value. Set the length of the track to how long the character took for switching lanes. The length of the track was set as 0.2. Right clicked and added a key. Set the time to zero and the value to zero. Right clicked again and set the time to the end of the track and set the value to 1. Closed the editor.

Dragged the main output pins of the set nodes of both the left and right region and pinned it to the play from start pin of the timeline node.

Once the timeline is finished the player’s current track changes to the previous new track value. So, drag the finished pin from the timeline node, insert a set Lane node and to this plug in the NewLane value.

During the execution of the timeline the player movement needs to be updated in real-time. So, from the update pin, extend it and add a SelectActorLocation node. No need to set the actor as we are in the actor class and the default value of the node is self. The x and z co-ordinates are not different from the previous state. But they are still changing as the player moves forward so dragged in a reference of the capsule and split the location of that node as well as the SelectActorLocation node by holding down alt + right click and selecting the spit struct pin option. Plugged in the x and y values.

For the y component now we have the lane no: and the location corresponding to the lane number is in the LaneY variable so get a reference of the LaneY variable add a get node and to one input plug in the array – LaneY and to the other input enter the initial lane number i.e. Lane node. Thus, we get our initial player location. To get the final player location, just add another get node, to one input plug in the array as we already have that node, just drag the output to this too. To the next input pin, enter the final lane number i.e. NewLane node. Now we have the final player location. To gradually change the values from the initial and final value, added a float lerp node. To the A gave the initial location i.e. dragged the output of the first get node into it. To B gave the final location. To set the pace of the transition to what we want it to be, extended the update node of the timeline into the alpha of the lerp and gave the output of the lerp to the y co-ordinate of the SelectActorLocation node.

While running spent a lot of time pressing A & D but it’s the arrow keys that trigger the whole thing.

Could eliminate the LaneY by directly using the location values but storing the real values in LaneY and using lane numbers is much easier while tweaking the program if needed and also computation is much faster as it’s in terms of 1 and not 330.

**Point scoring**

For each tile the player covers, his points get increased. Went in ThirdPerson Game mode and made an integer variable Score at set the default value as 0 and another integer variable points multiplier and set the default value as 1 otherwise if default value was 0, no points will be multiplied as multiplication with any number would only return 0. Went to the master tile class and went to the destroy tile section and shifted the 2 tiles to the right as we need the points to be increased before the tile gets destroyed. Inserted a cast to third person game mode node and set the object wildcard as get Game Mode node. Extended the as third person game mode and added a get score multiplier node from it added an integer\*integer node and to the other input set the number as the number you want to be multiplied (Here 1). Extended the as third person game mode and added a get score node and extended it and added an integer + integer node and hooked the integer\*integer node’s output as its input. From the third person game mode added a set score node and connected the integer + integer node as it’s input. Extended the exec pin of the cast to third person game node to the set score node and extended the set score node’s exec pin to the destroy tile section.

To display the score on the screen made a new UI blueprint class and named it ScoreHUD (Heads Up Display). Opened it and inserted a text box onto the screen in the right-hand corner. In the appearance section changed the size and font to the appropriate value. Adjusted the text box according to the size of the font. Pressed the anchor button and anchored it to the top right-hand corner. Set the default text as 0 (Just in case so that the user doesn’t in any way see Text written on the screen). To make the text dynamic i.e. to make it display the score variable’s value went to the content and added a binding by clicking the bind button and clicking the create binding option. An event graph appears and extend the get text node and added a cast to third person game mode node. Set its object wildcard as the get Game Mode node and from the as third person game mode option extended it and added a get score node and plugged its output to the return value. As the score is an integer an integer to string node appeared and tweaked its parameters and unchecked the grouping as I didn’t want commas to appear. Also set the minimum and maximum values.

To get the widget blueprint to be displayed on the screen, went to third person character class and added a begin play node and extended it and added a create a widget node and set the widget class as ScoreHUD and extended the return value and added an add to viewport node.

**Increasing speed as game progresses**

We want the character to increase his speed as the game progresses. The speed of the character must increase as he covers every tile. Went to the master tile class and dragged the destroy tile section to the right and in between added a cast to third person character node and set its object wildcard as the get player character node. Extended the as third person character pin and added a get character movement reference which can be found by scrolling to the end and extended it and added a set max walk speed node. To set max walk speed as a multiple of its previous walk speed, extended the get character movement reference and added a get max walk speed node and extended it and added a float \* float multiplier node and to the other input set it as 1.02 to make the transition gradual. As we don’t want the value to go beyond a certain number, inserted a clamp node of float type. The min. had to be set as 600 as that was the default max. walk speed, the maximum was set as 1200 for now (Max. walk speed doesn’t mean that that is the speed the player walks it only means that it is the maximum speed a player can attain). Plugged in the output pin to the set value of the set Max walk speed node. Plugged in the exec pin of the cast to third person character to the set Max walk speed node and plugged its exec pin to the destroy tile section.

**Spawning pipe obstacles**

Made a pipe in blender by deleting half of a torus of less sides. Then selected half the portion of the torus and moving it to one side, leaving the centre loop in the middle. Then scaled down the centre loop to make it look uniform and exported it as .fbx.

Dragged in the .fbx file into the content browser. In the box that appeared, checked the import skeletal mesh and also checked the import all option and the mesh was created. Opened it and added a simple collision. Created a material and added a constant 3 vector and hooked it to the base color option and set the color as red. Added a constant vector node and hooked it to the metallic option and set an appropriate value. Made a blueprint class of actor type and dragged in the pipe mesh into it.

In the master tile class made a new function and named it SpawnObstacles in the function graph extended it and made a switch node of integer type. To its input plugged in a random with limits node and set the min. as 0 and max. as the value depending on how random and often the spawning had to take place. When the input of the switch was 0 I wanted it to spawn a pipe on lane 0, when the input was 1 I wanted it to spawn a pipe on lane 1, 2 on lane 2 and 3 on all 3 lanes and so on. So added the necessary number of pins and extended pin 0 and added a spawn an actor from class node and set the class as pipe class and in the set transform input of it as the position of Lane 0 was stored using an arrow named Lane0, dragged in a reference of it and extended it and added a get world transform node (as Lane0 was an arrow and co-ordinates were needed), extended its output and plugged it in to the set transform option.

Did the same to pins 1 & 2 only difference was instead of Lane0 the respective arrow references were used. For pin 4 made a copy of the previous 3 ones and hooked 0 to 1 to 2 so that pipes were spawned on all 3 lanes.

Next went to the event graph of the master tile class and dragged in the SpawnObstacles function node and connected it.

**Spawning Tank obstacle**

Unlike the pipe obstacle, the tank obstacle turned out to be a bit troublesome, had to make a tank in blender by adding a plane and then increasing the loop-cuts along its length and breadth. Extruded it to get the top and extruded sets of 4 squares along the corners to get the stand. Duplicated the stand region in order to be able to give it another colour. Increased the loop-cuts on the faces on the tank and selected faces so as to form an HP and duplicated it so as to give another colour and exported it as .fbx.

On second thoughts could have extruded the plane straight away and only made loop cuts on the sides I need to reduce the number of polygons but it was a lot easier this way and now big impact on performance seen so far.

Dragged it into the content window and checked the skeletal mesh and imported all but a whole lot of files appeared. Deleted all the physics files but turned out it was needed to open the main tank object so created one automatically but still in the object viewer appeared different. Only options for adding skeletons and animations were there however there was a create skeletal mesh option so clicked on it as that was what I needed after all. The skeletal mesh was created and I opened it and in the object viewer I checked the complex collision option. Made a blueprint class of actor type and dragged in the tank skeletal mesh. But still collisions didn’t work during run-time. So inserted a box collision and placed it just covering the top and set the collision type to block all. This did the trick and the player couldn’t run through the top. Could’ve added boxes for each leg but as the player can’t run wherever he wants and only along the centre of each lane it was better not to increase the complexity unnecessarily.

To spawn the tank obstacle was simple just made the same nodes like the pipe obstacle. Only in this case the actor class was set as the tank class. However, the transform and the randomness were a problem. Could’ve added the pins on the previous switch node but instead of it the lane 0 pipe obstacles’ last node’s output pin were extended and a switch node was added and the functionality for spawning the tank actor was added. Did the same for the other 2. However, if the transform Lane0 was used for the tank actor connected to lane 0’s pipe actor then it would lead to cases where both spawns at the same time and the player can’t jump or duck. Solved the problem by using Lane1’s transform for the tank connected to Lane0’s pipe obstacle, used Lane0’s transform for the tank connected to Lane2’s pipe obstacle and so on.

This led to the spawning of pipe on one lane and tank on the other which is a whole lot fun.

**Coin Pickup**

The goal was to spawn coins on tiles with no other obstacles and to make a sound once they were collected and also display the number of coins collected on the screen.

So the logic is as soon as the player overlaps or collides with the coin it should play a sound and destroy itself and the coin count must get incremented.

In blender made a coin shape using a circle of fill type using the option above nygon and made knife cuts in a circle and deleting the edges I didn’t want. Exported it as a .fbx file. Dragged it into the content browser and made a material. Used a constant3 vector and gave it a yellow color and plugged it into the base color. Copied the constant3 vector node and inserted a constant1 vector node and plugged but into a multiplier node and hooked its output into the emissive node and adjusted the constant1 node’s value to control the emission.

Went to third person game mode class and made a new variable CoinScore of integer type and default value 0.

Opened the imported coin file but as a mesh wasn’t created made a skeletal mesh by clicking the button. Opened the skeletal mesh and added the material and also added in a complex collision. Made a blueprint class of actor type and dragged in the skeletal mesh. To make the object rotate in the scene added a rotation component. The default settings of the component were enough to make it rotate along the axis. Clicked the mesh’s on actor begin overlap button and went to the event graph with that node created and extended the other actor pin and added the cast to third person character as that is the actor that should trigger the overlap. Extended the exec pin and added a cast to third person game mode node and set the object wild card as the get game mode node. As third person game mode is connected to the get CoinScore node and extended it to the integer increment node. Also connected the third person game mode’s exec pin to the increment node. Extended the as third person game mode and added a set CoinScore node and set its value as the output of the increment node. To the exec pin of the increment node added a play sound 2D node and set a sound and extended it and added a destroy actor node. As the node points to the coin actor by default no need to set it.

To spawn the coins, went to the master tile class and added a spawn actor from class node and set the class as the coin class. To ensure that the coin appears in a lane where there are no other obstacles in the first row if the pipe was given Lane0 and the tank Lane1, the coin was given Lane2 and so on.

Went to the scoreHUD class and added a new text box and adjusted its location and anchored it to the top-right hand corner just below the main score and set the default text as 0 and aligned it. Added a binding and dragged the last node and between it added a cast to third person game mode and extended the as third person game mode and added a get CoinScore node and hooked it to the return value and also disabled grouping.

The coins spawned nicely but the begin actor overlap part didn’t work so went back and there was a problem with the static mesh, the simulate physics wasn’t working so added a box collision object and transformed it so that it just covered the coin and then clicked the box collision’s on begin overlap button and went to the event graph and replaced the static mesh’s on actor begin overlap node with the box’s on actor begin overlap node and extended the other actor pin to the cast to third person character’s other object wildcard.

**Ragdoll Death**

We want the player to have a ragdoll death whenever it touches an obstacle. So, the logic is this: make a ragdoll simulation and a function which triggers this simulation and after that pauses the gameplay. The function should be triggered whenever the actor overlaps with an obstacle/collides with an obstacle. We use overlap as it’s the easiest.

Went to the third person character’s skeletal mesh. Although it had a physics asset wanted to make one by myself so right clicked and clicked the create and assign option in the physics asset so that the physics gets automatically assigned to it. Opened the static mesh and clicked the generate all bodies button at the bottom and expanded the toolbar on the top and pressed the simulate button to view the ragdoll animation.

Went to the third person character class and created a new function called Ragdoll. Extended it and made a set simulate physics and the target is mesh. Extended it and made a disable movement node whose target is character movement and extended it and made a set game pause node. To test it in the event graph went to input and, in the keyboard, section added the C node and extended the when pressed pin and connected it to the Ragdoll function. But when tested, the game is paused before the animation so went back and deleted the pause node and now collision happens but the player falls through the scene so went back and set the character mesh’s collisions to block all dynamic. Now whenever the C key is pressed the Ragdoll death happens. Made sure that the player is still able to pick up coins and get blocked by obstacles.

To make the player do the ragdoll death whenever he’s hit by an obstacle, went to the pipe actor class and in the event graph extended the event actor begin overlap and extended the other actor and added a cast to third person character and extended the as third person character and inserted the Ragdoll node. Changed the pipe object’s collision to overlap all dynamic/overlap all.

Did the same for the tank obstacle. But in case this doesn’t work go to the box collision of the class and click the on actor begin overlap and do the same process. Whether you are using box collision or the actor overlap make sure its collision is set to overlap all dynamic/overlap all. Only then will the player be able to pass through it and will the event be called.

**Game Over Screen**

The logic is to create a game over widget and to call it whenever the ragdoll function is called so just had to call it in the ragdoll function.

Just created the game over template in this step. The functionality will be given later. Made a widget blueprint and named it GameOverScreen. Opened the designer and placed a text box and anchored it to the left side and gave the text Game Over, similarly added ‘score:’ and ‘coin:’ text boxes. Also added Restart and Main Menu buttons. Gave all of them a yellow colour for some reason the eyedropper wasn’t working. Compiled it and went to the character blueprint and went to the Ragdoll function graph extended the disable movement node and made a create widget node and set the class as GameOverScreen and extended the return value and added a add to viewport node. Compiled it and run it but an error was shown which basically said that it can’t run in the editor when editor is working with colors. Just saved current to solve this. After this the game could be played.

**Adding Pause Button Functionality**

The aim was to make a pause button on the top-left corner of the screen which when pressed shows the Resume and Go to Main Menu option. So the logic is to make a button in the ScoreHUD which on clicking closes the present viewport and takes the user to a new UI screen with Resume button and Go to Main Menu option.

Too a screenshot of Youtube’s pause button and dragged it into the content browser. Opened the ScoreHUD and made a button on the top-left hand corner and anchored it to the same position and uploaded the pause button image in its place. Scrolled down the button properties and clicked the on click button and the editor graph with the on click node is shown. Extended it and made a set game paused node and checked the pause option to ensure that the game is paused and extended it and added a create widget node and extended it and added an add to viewport node. Couldn’t compile it at this stage as a pause game widget didn’t exist. So made a new widget blueprint (option was in the UI section) and named it GamePaused. Opened it and added a text box at the centre and anchored it to the centre and set default text as “Game Paused” and gave it a yellow color. Dragged in two buttons and placed them side-by side below the text and anchored both of them to the centre of the screen. Inserted text boxes in them and set default text as “Resume” and “Go to Main Menu” respectively. Gave the buttons a blue color and the text a yellow color. Selected the resume button and in its properties section clicked the when clicked button and in the event graph extended the when clicked node and added a set game paused node and made sure the paused option was unchecked so that the game resumes and extended it and added a remove widget’s remove from parent node (typed in remove widget and 2 options came remove all widgets and remove widget from parent). This is to remove the widget from the screen when the game resumes. Note that the game resumes and then the pause screen goes there is a small time-delay, almost negligible but keep in mind to reverse the order in crucial games but I don’t know whether that setup will work.

Opened the ScoreHUD and set the widget class in the create widget node as GamePaused class. Now whenever the pause button is clicked, the Pause menu comes and whenever the resume button is clicked the game resumes.