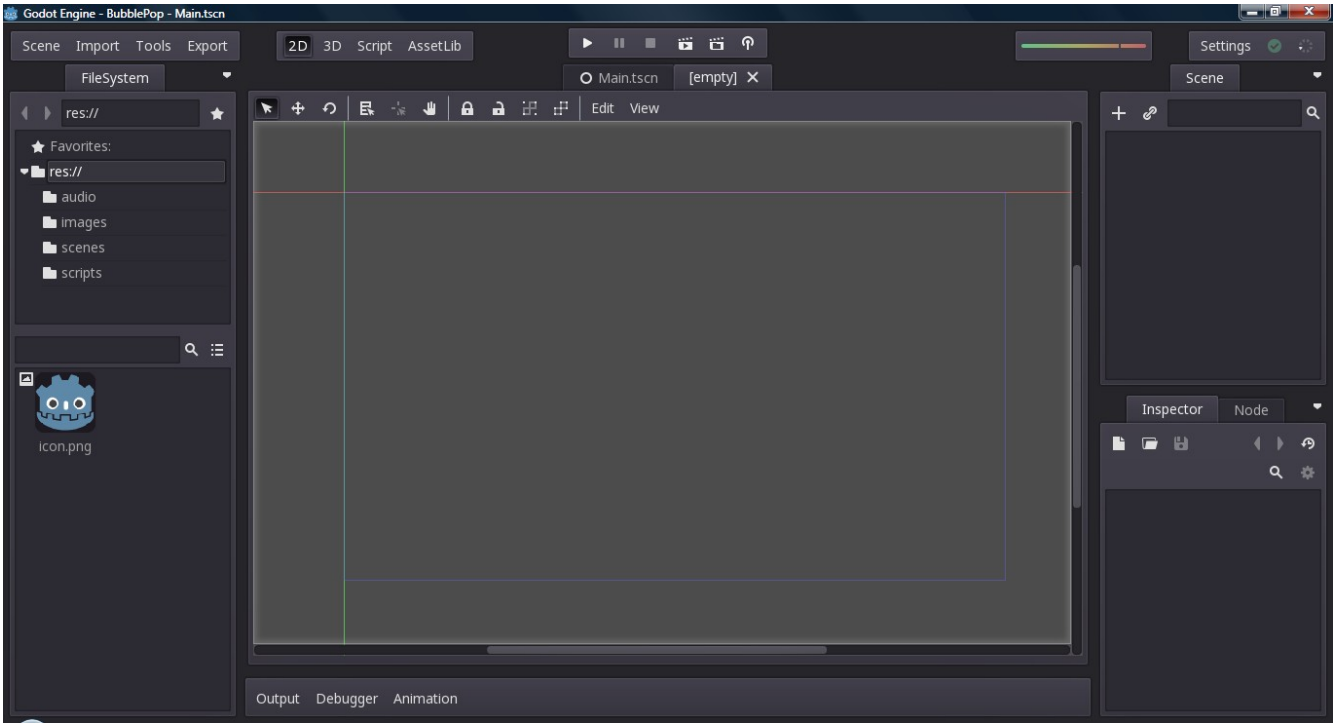
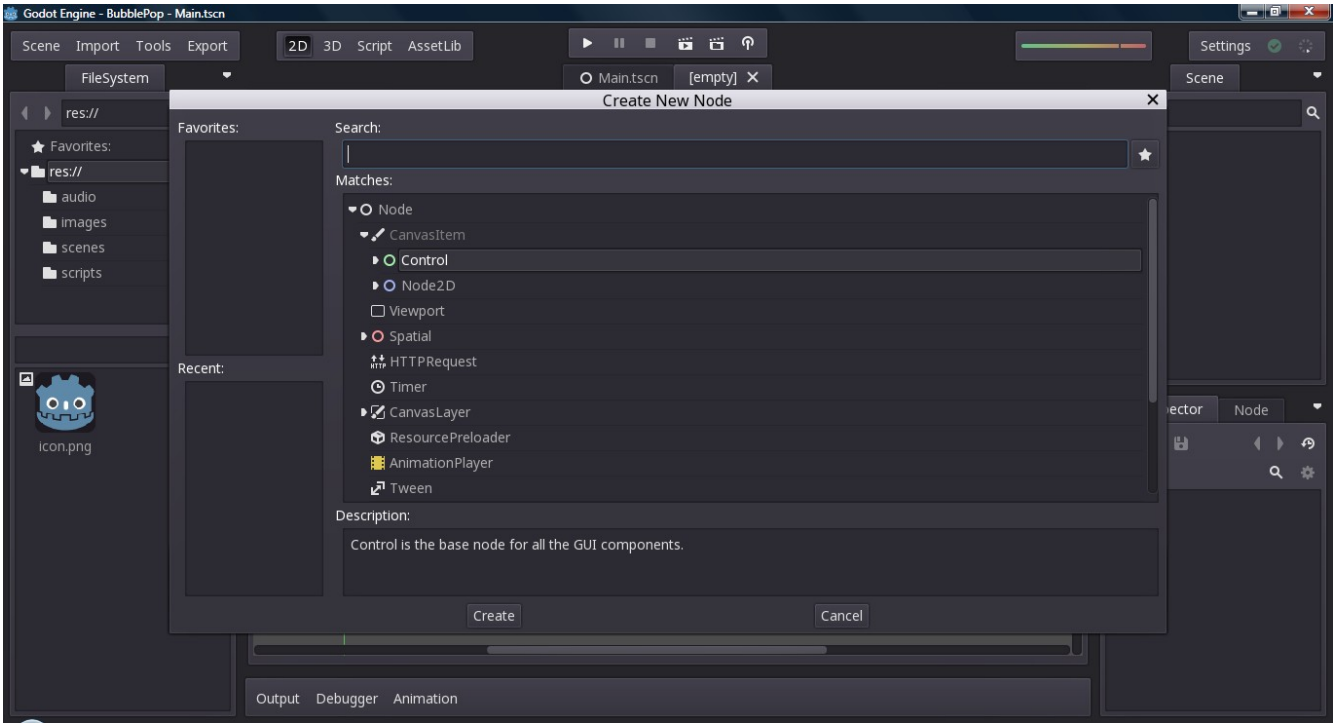
Godot 2D Game
Lesson 12: UI Programming

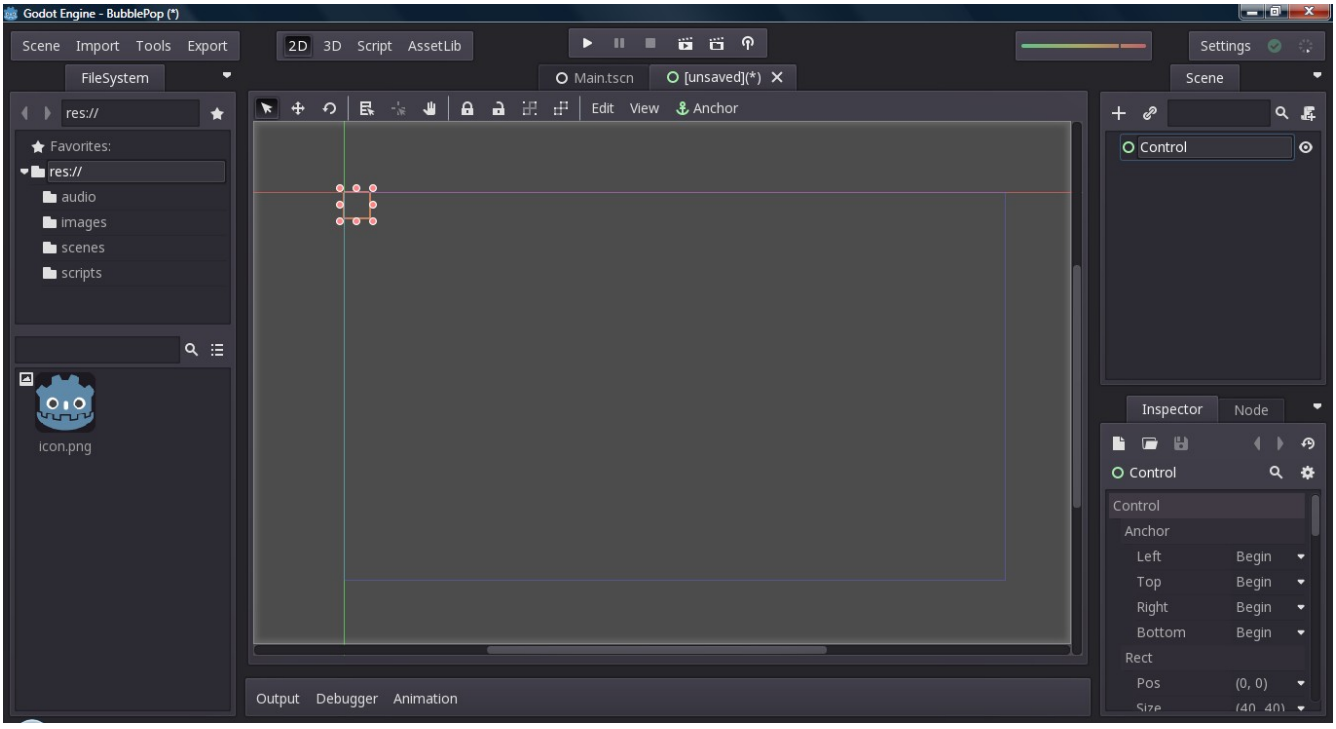
In this lesson, we are going to begin designing the UI for our game. We will start by creating a new scene:



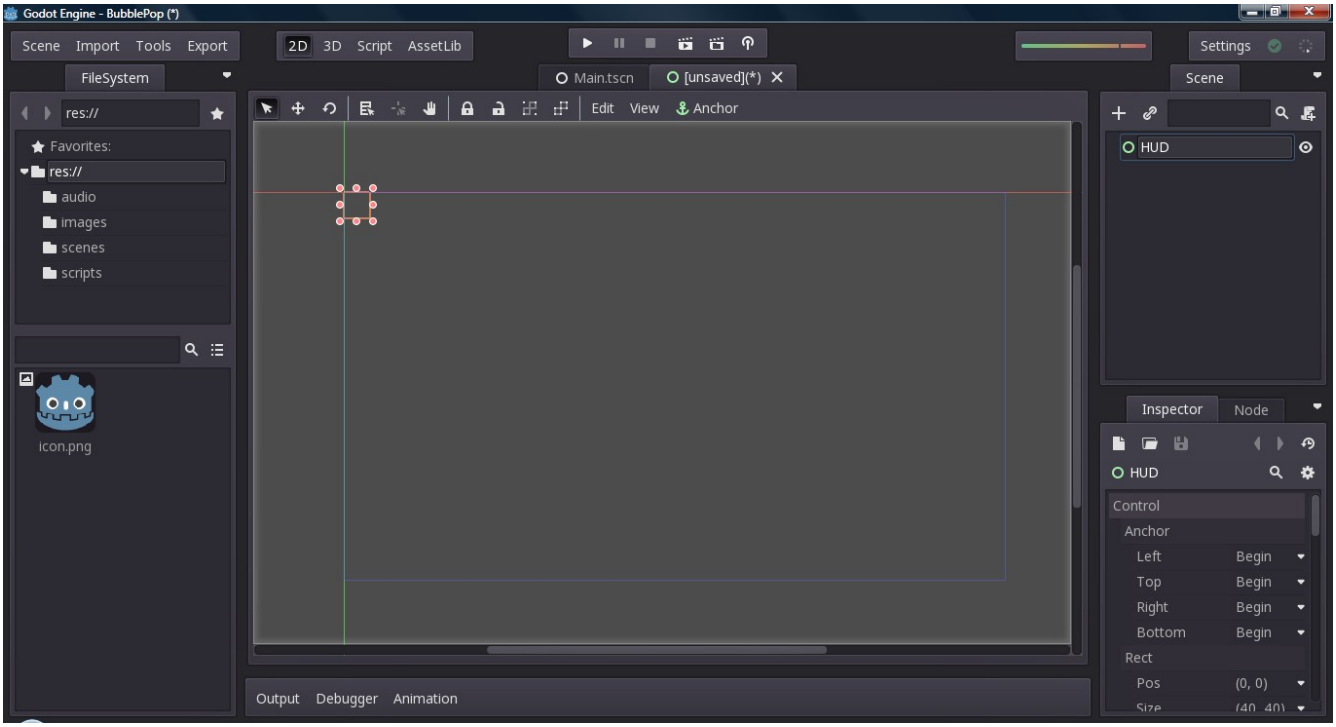
Next, we will click the plus sign to create a new node:



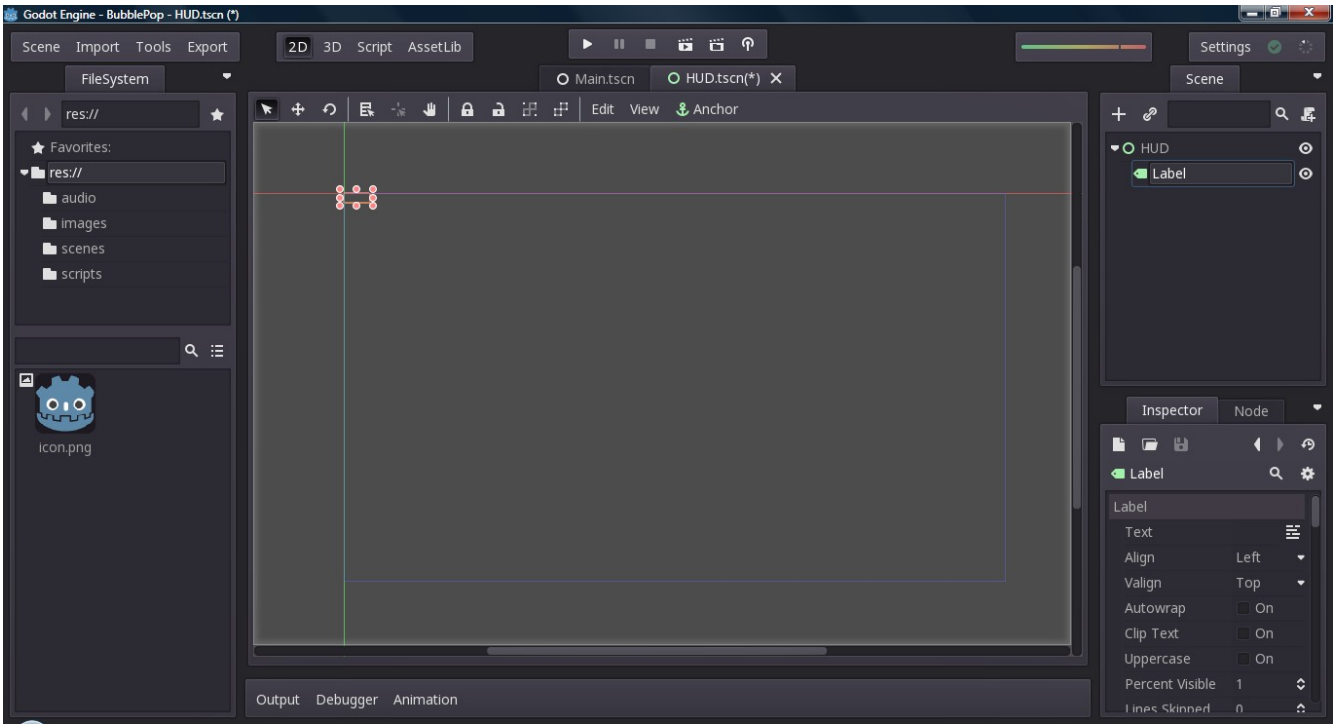
Choose the Control node and click “Create”:



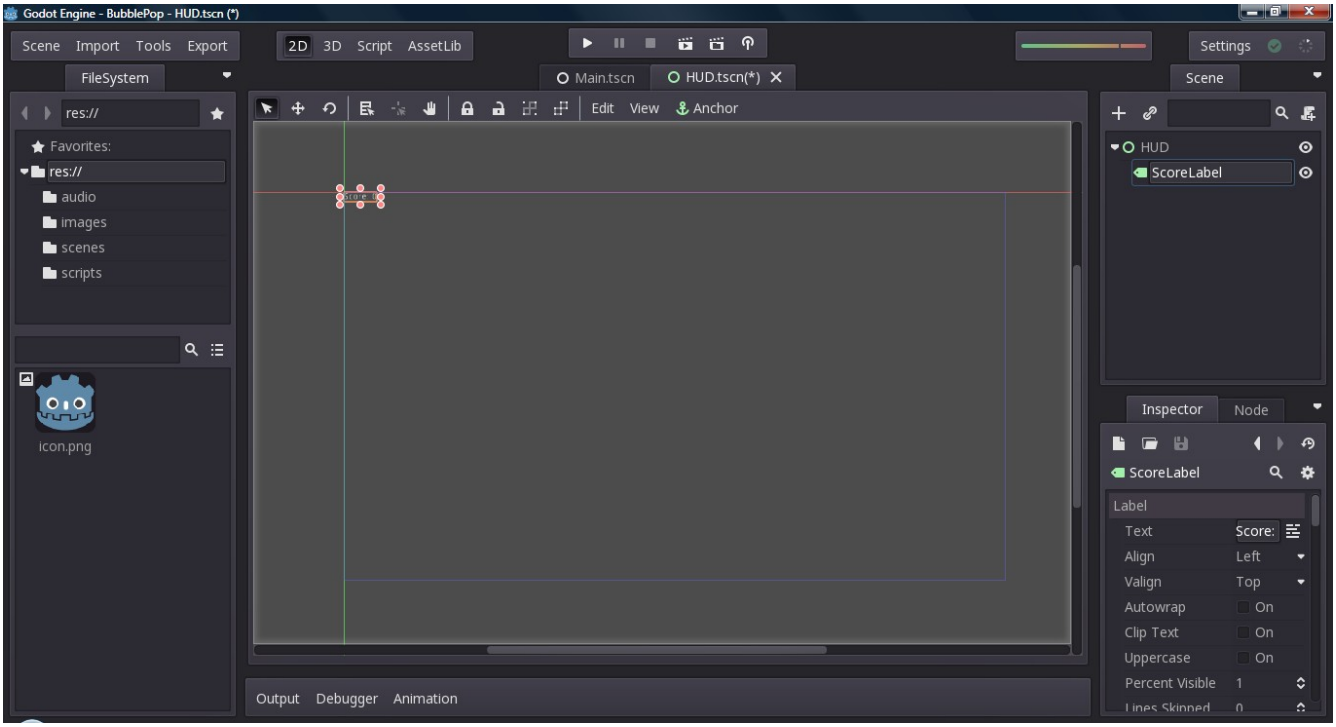
Now change the name of the new node to “HUD”:



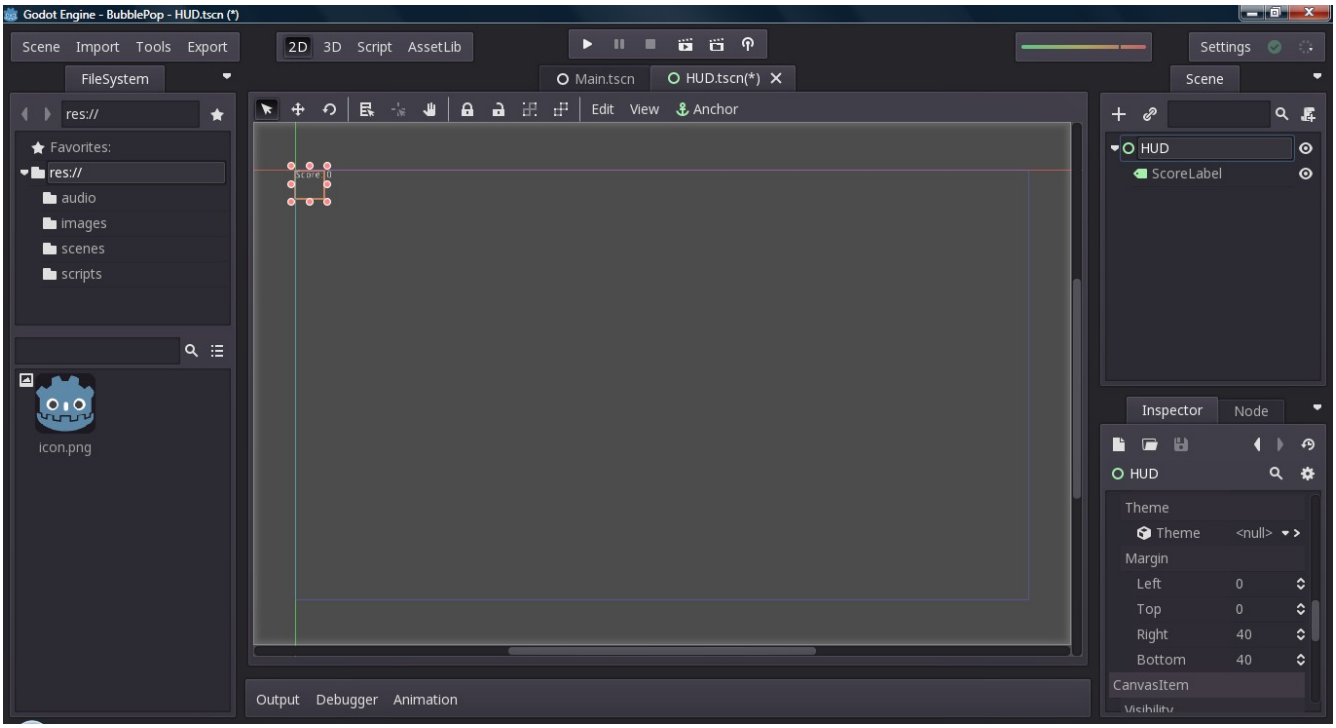
A generic control node works great as a container for other control nodes. Save the scene in your "scenes" folder. Next, we will create a new Label node:



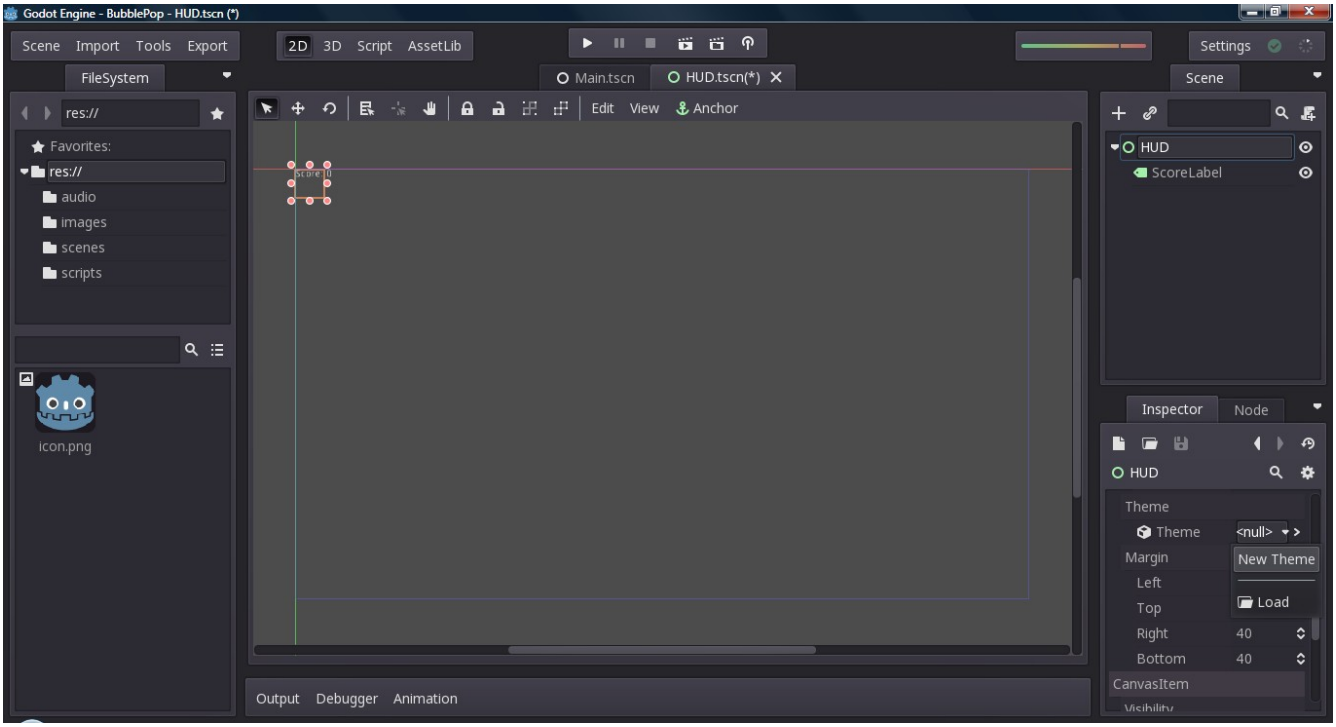
A Label node is used to display static text. Let's change the name of the label to "ScoreLabel" and set its "Text" property to "Score: 0":



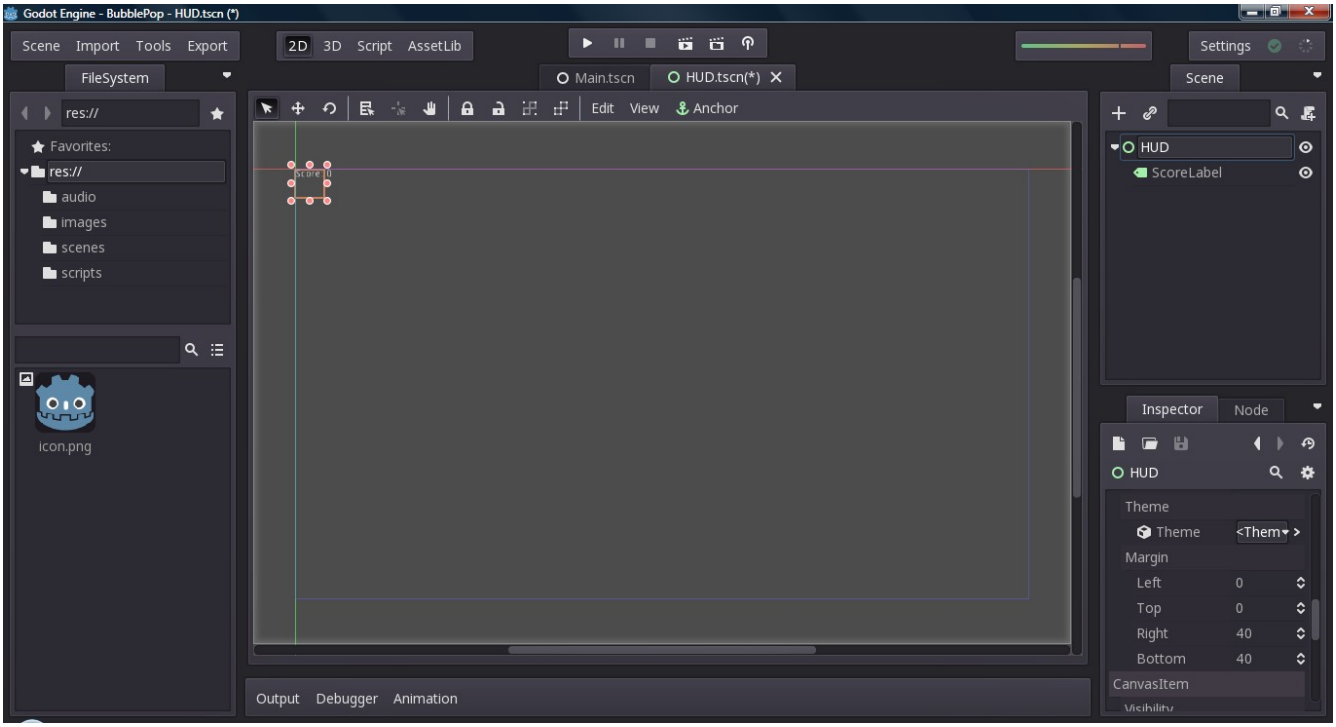
Wow. That text is quite tiny… How can we fix it though? Select the “HUD” node and scroll down until you see the “Theme” property:



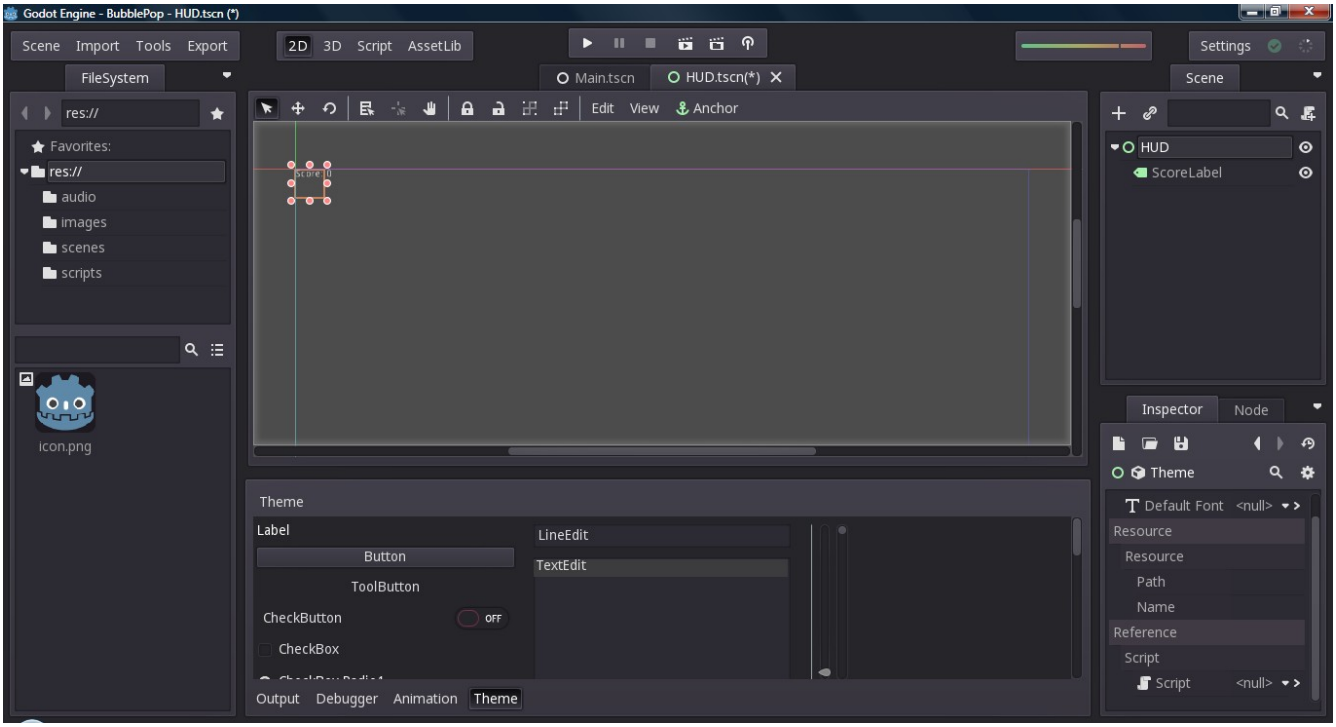
Now click the box beside the “Theme” property:



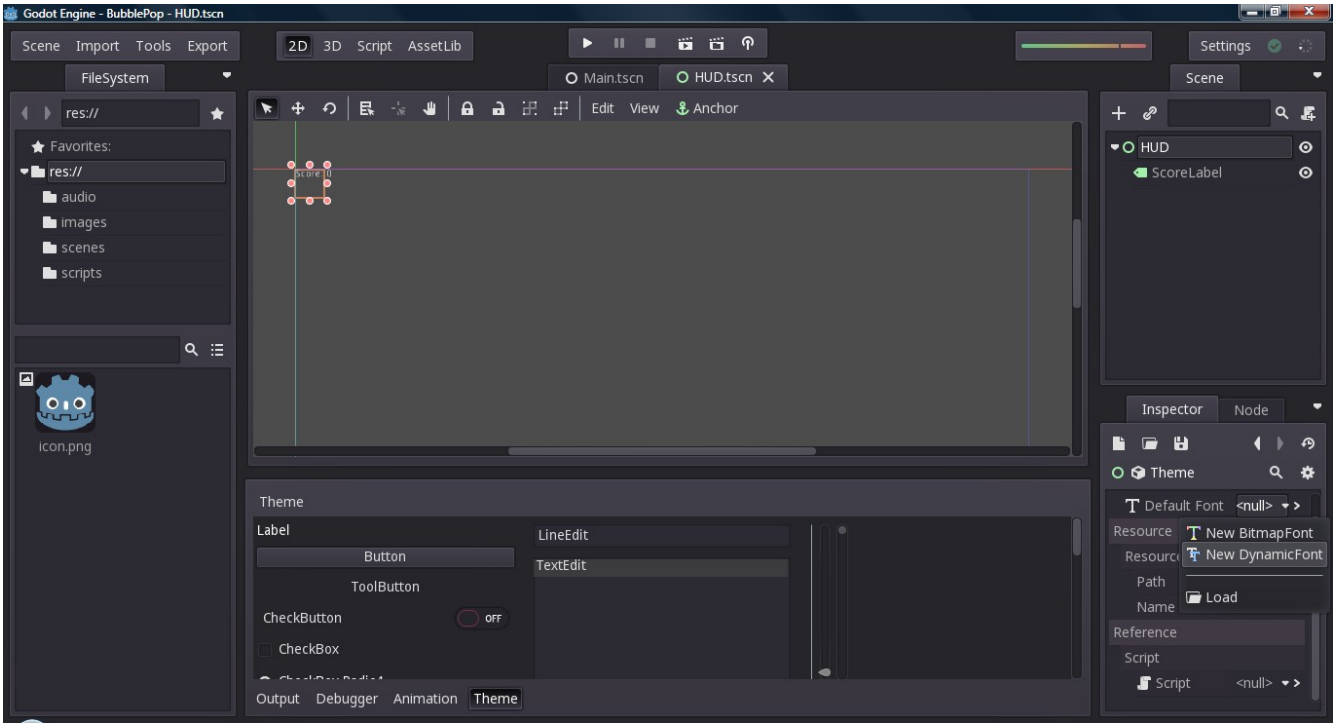
Click “New Theme”:



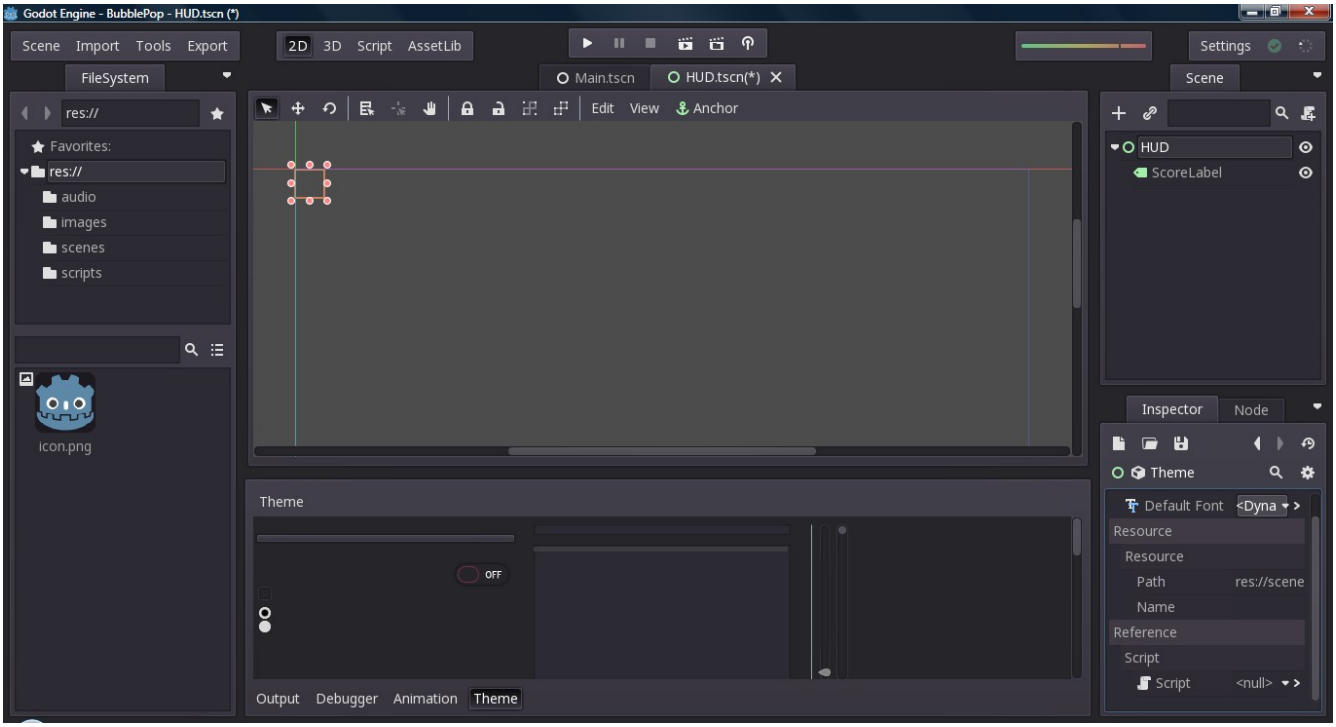
Now click the arrow next to the box:



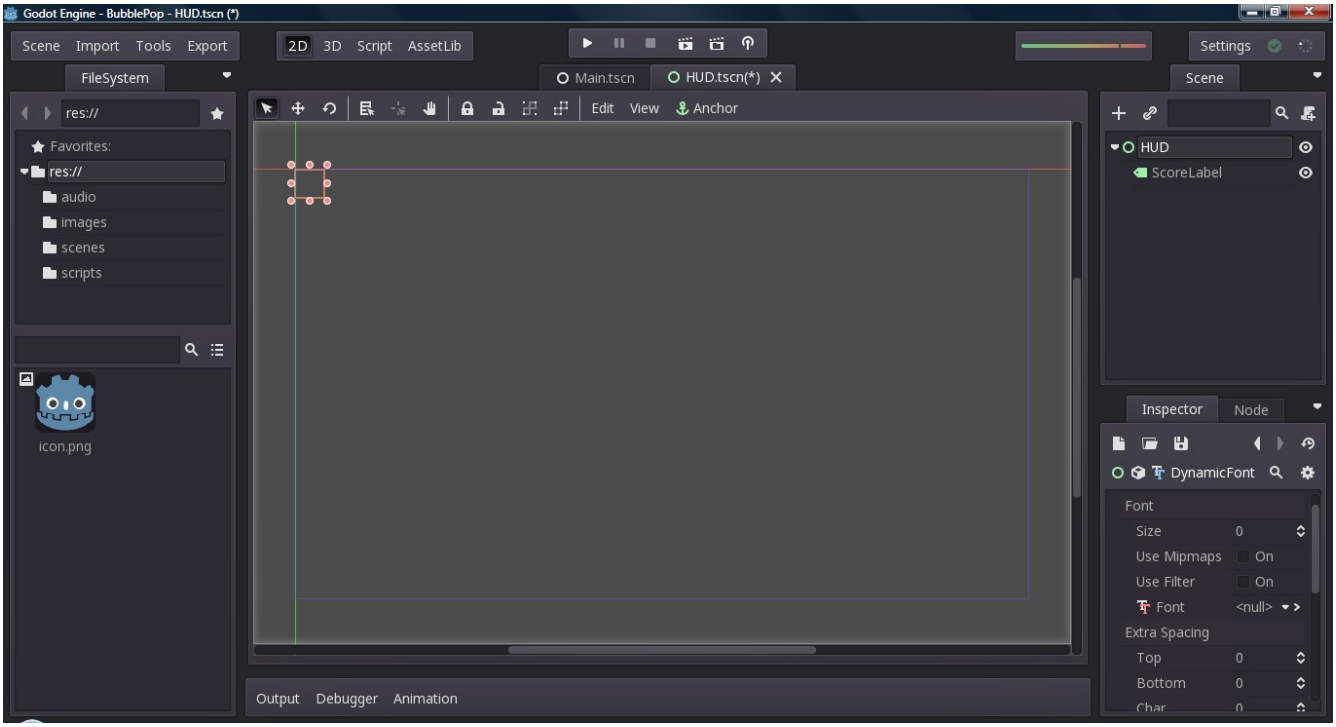
As you can see, we now have a bottom pane that shows the current appearance of the UI elements. If you look over at the lower right pane, you will see the "Default Font" property. That property is used to set the default font of all UI elements that are a child of the node we assigned this theme to. Click the box beside the "Default Font" property:



Now choose "New DynamicFont":



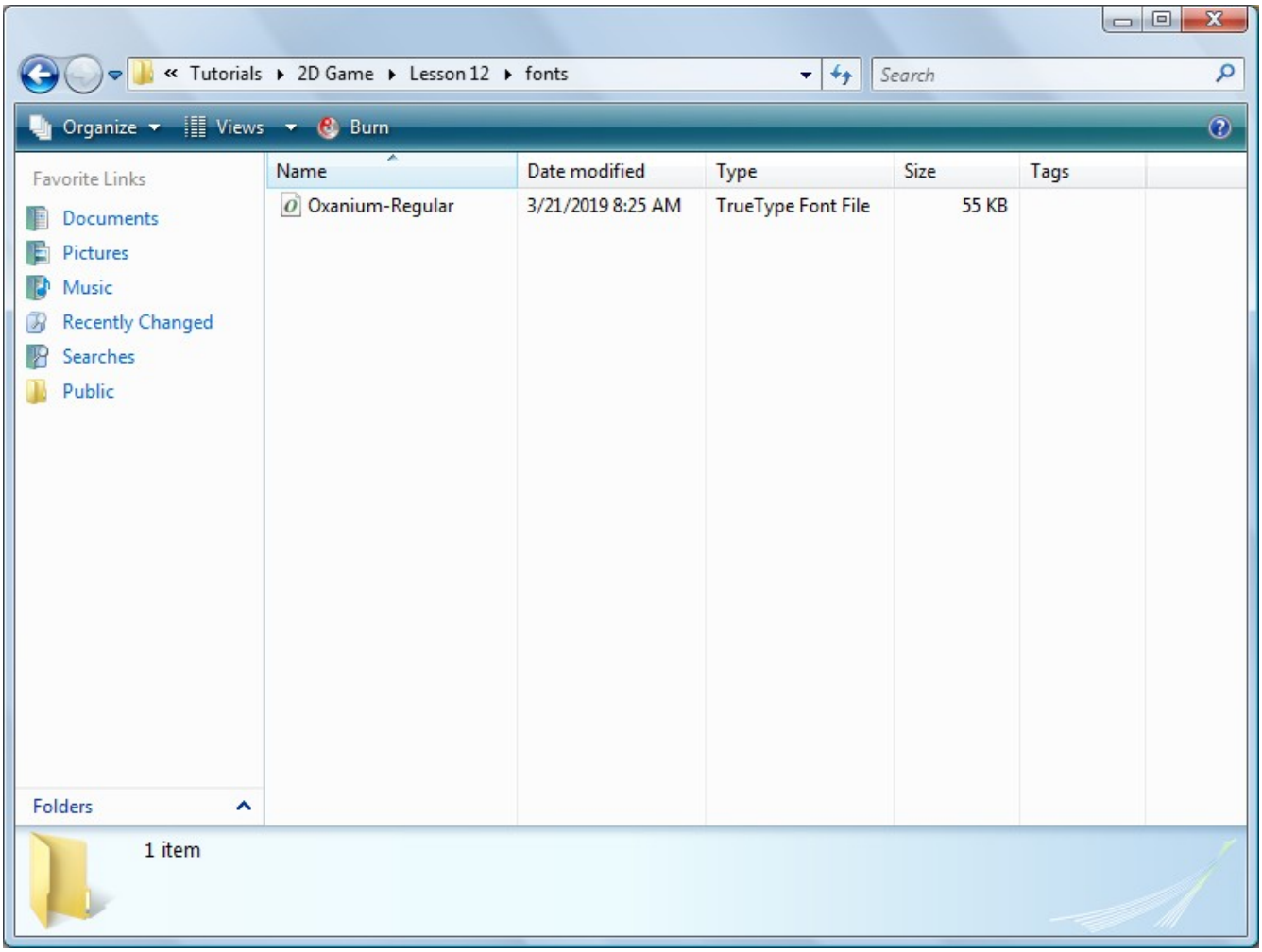
Huh… Why did all the text in the preview disappear? It disappeared because we need to choose a font file and font size before text can be rendered for the UI elements. Click the arrow next to the box:



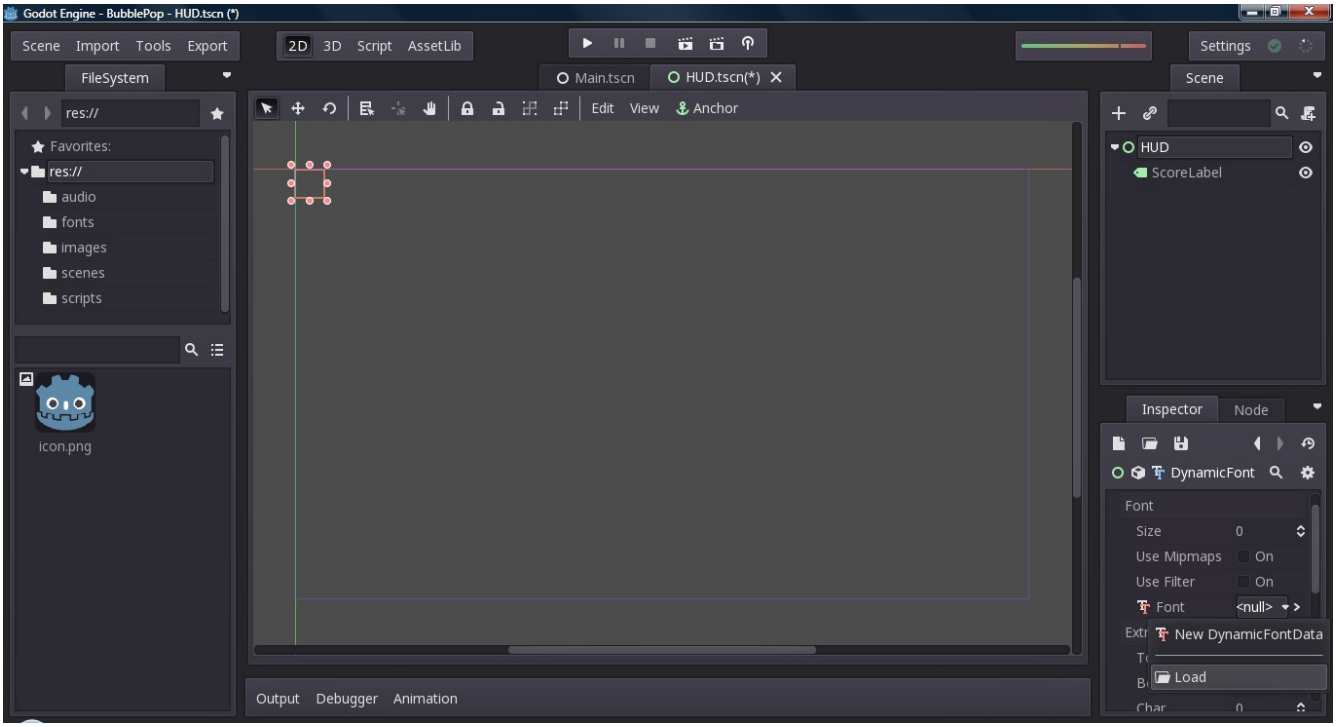
Before we can set the font, we need a font file. For this lesson, we will be using this font file:

[link to font file]

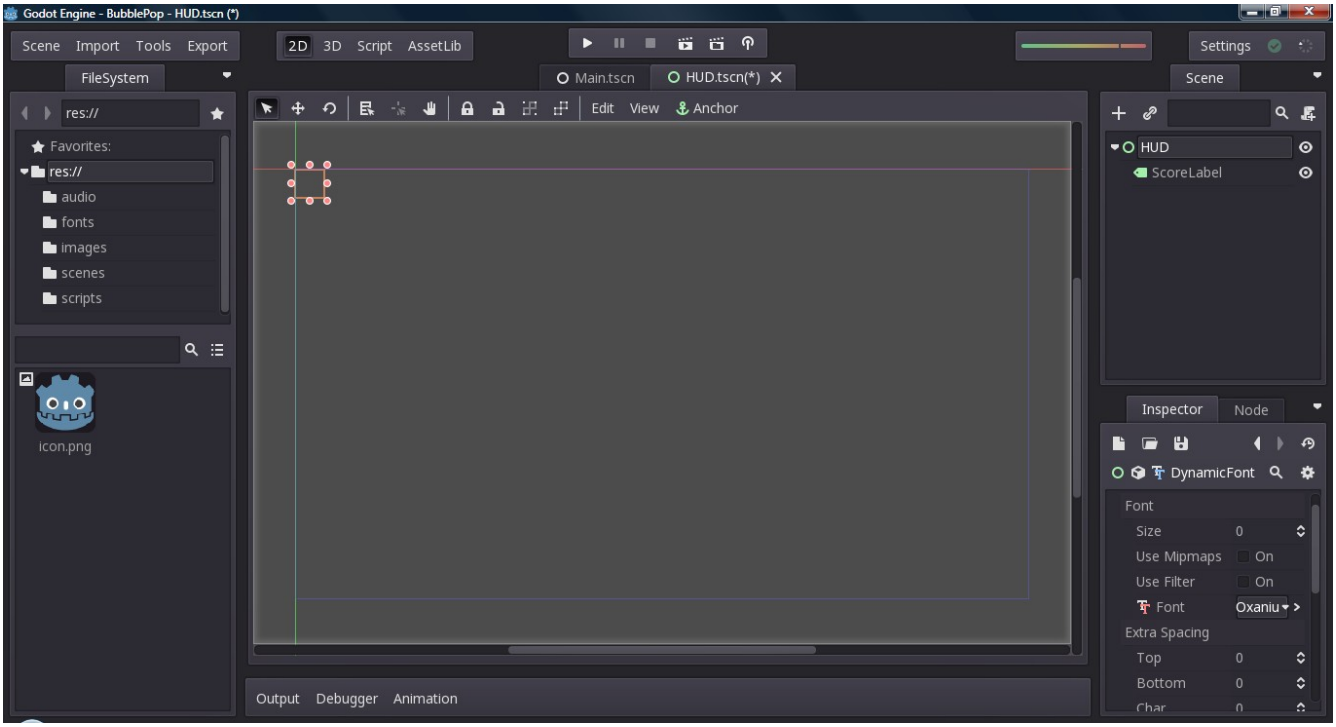
Create a new folder called “fonts” inside your project folder and download the above font file into the “fonts” folder:



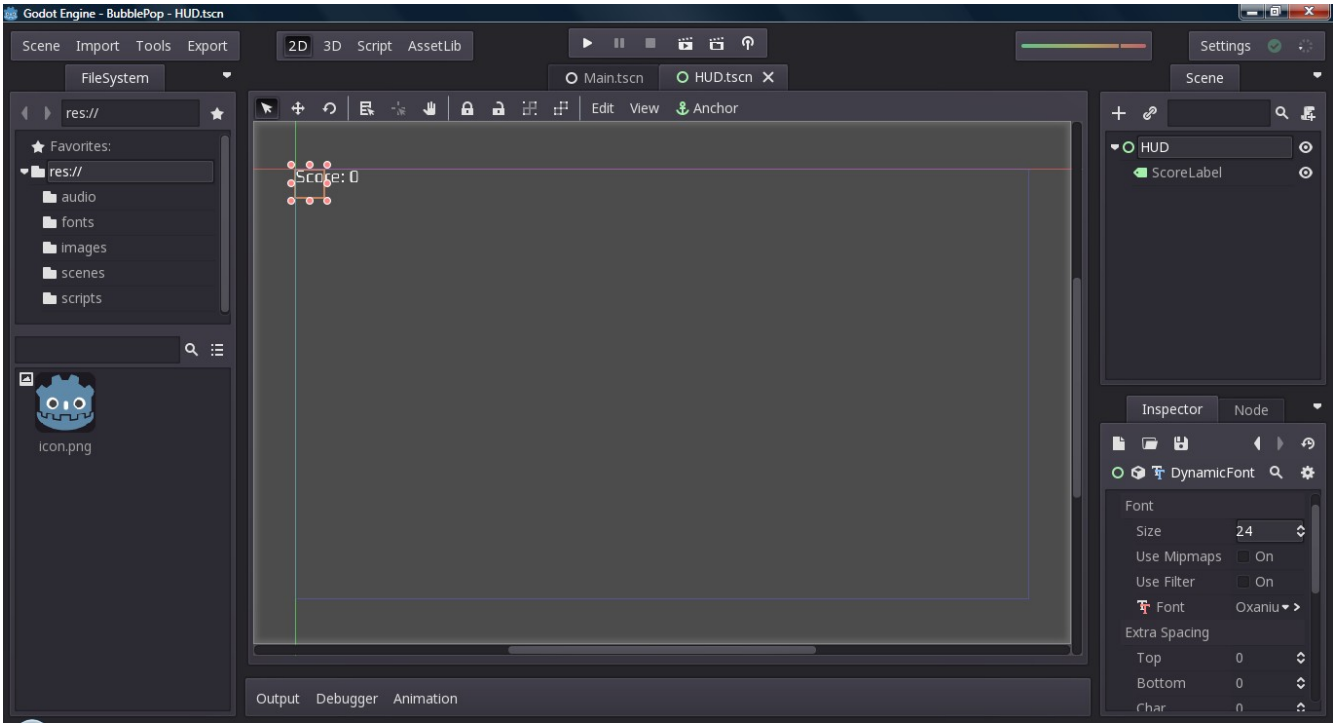
Now go back to Godot and click the box beside the “Font” property:



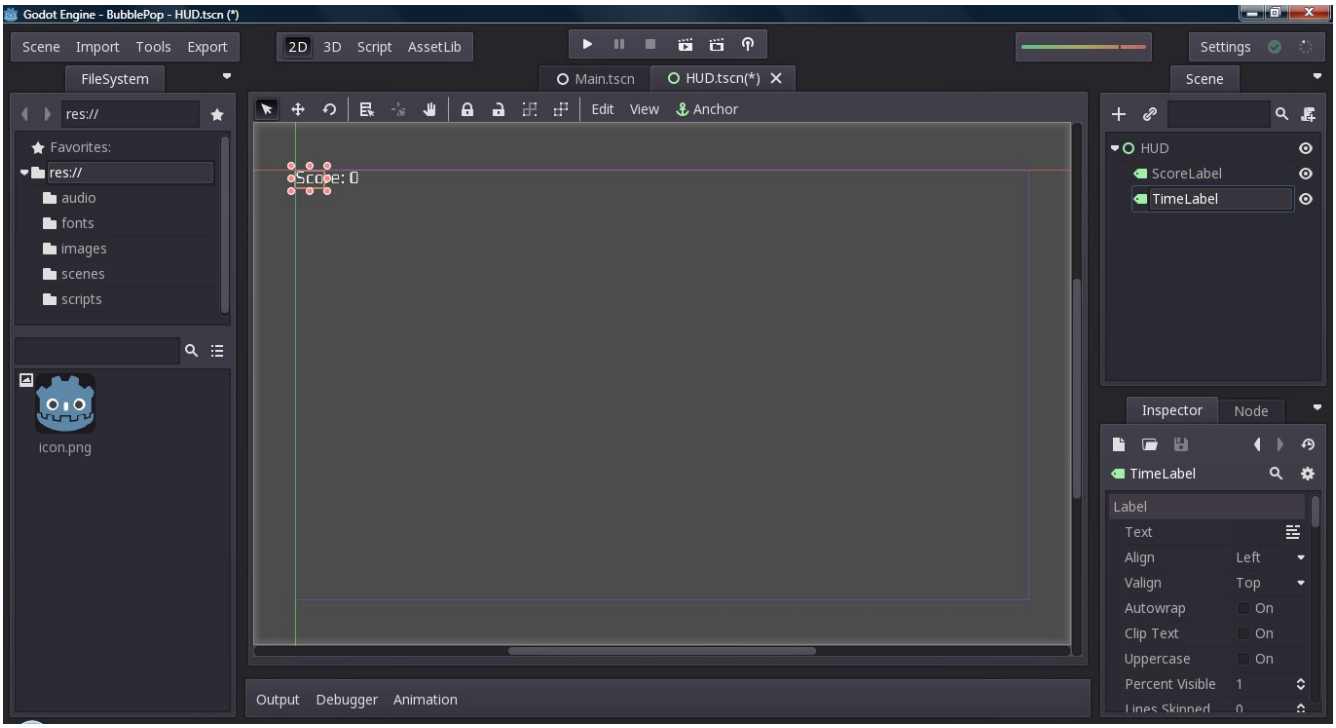
Choose “Load” and select the font you just downloaded:



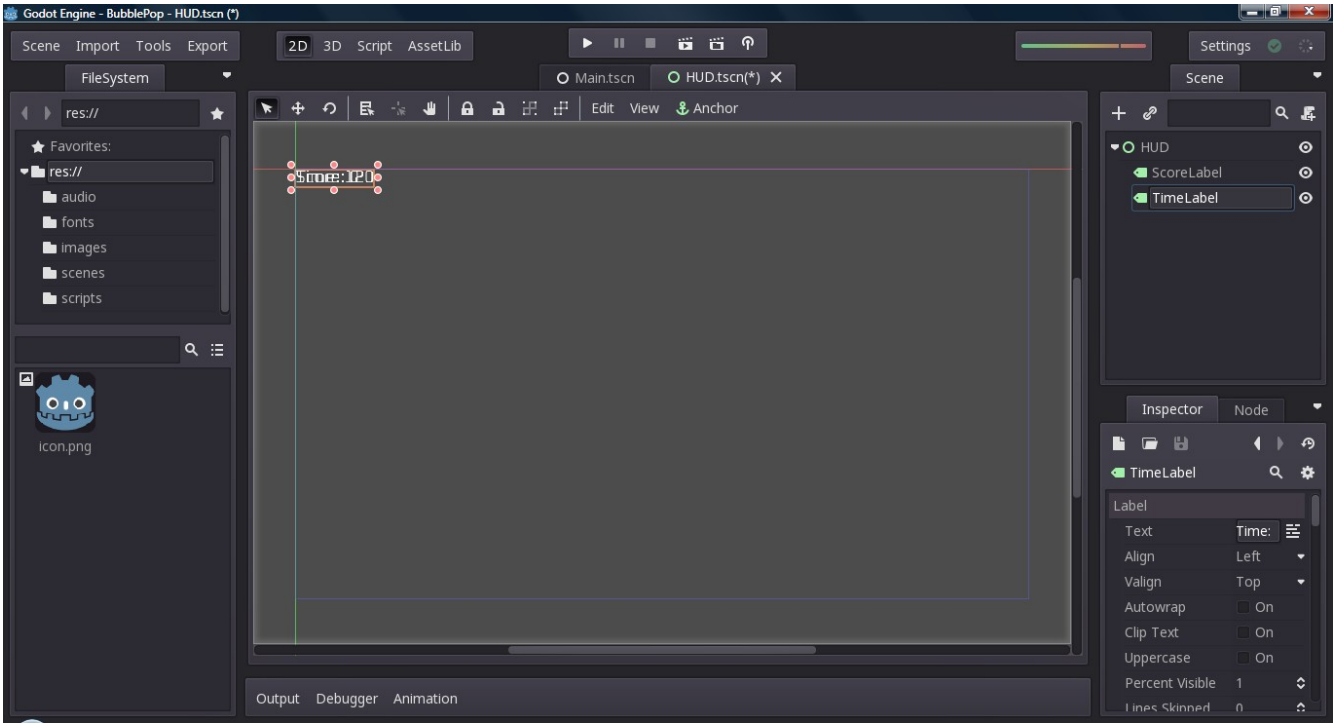
Now set the “Size” property to 24:



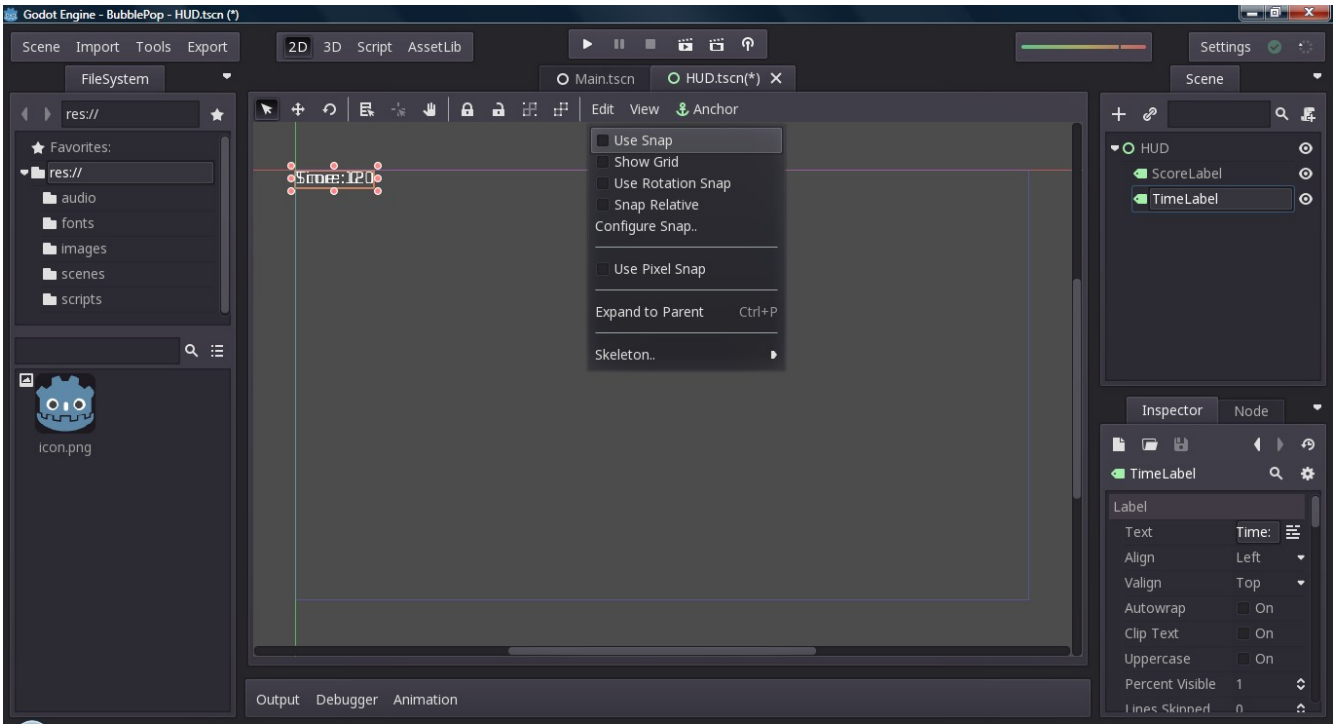
We now have larger text for our label. Let's go ahead and create another label called "TimeLabel":



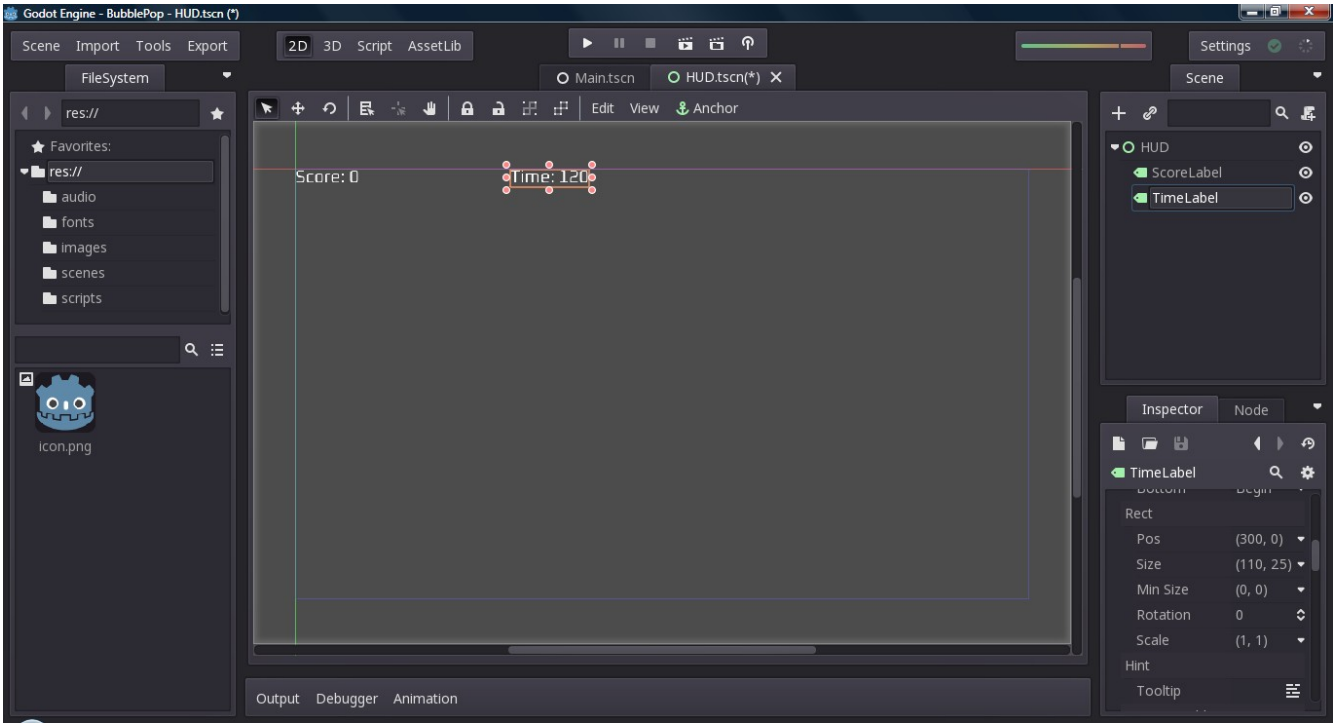
Now set the "Text" property to "Time: 120":



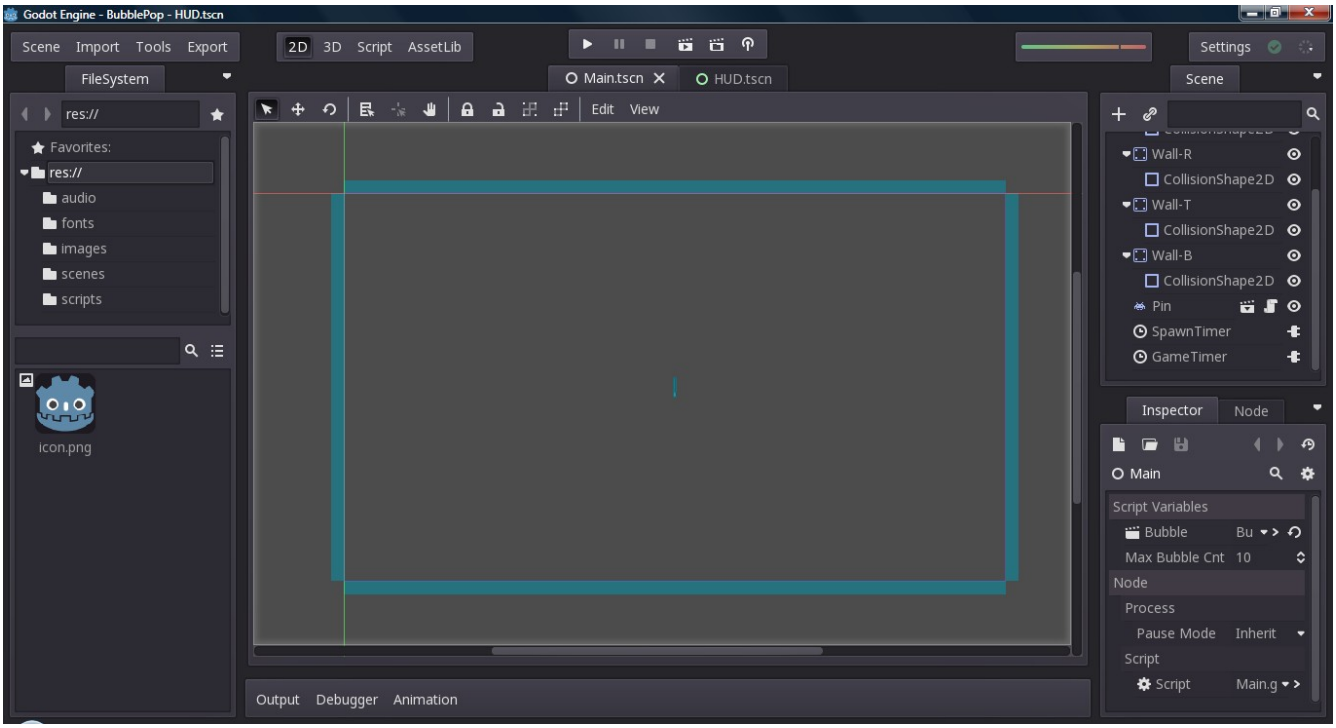
Hmm… the text is overlapping. We need to move our new label. But first, click the “Edit” menu and turn on “Use Snap”:



Snap will cause controls to snap to an invisible uniform grid when we drag them inside the middle workspace. Let’s drag our time label over a bit so that it no longer overlaps our score label:

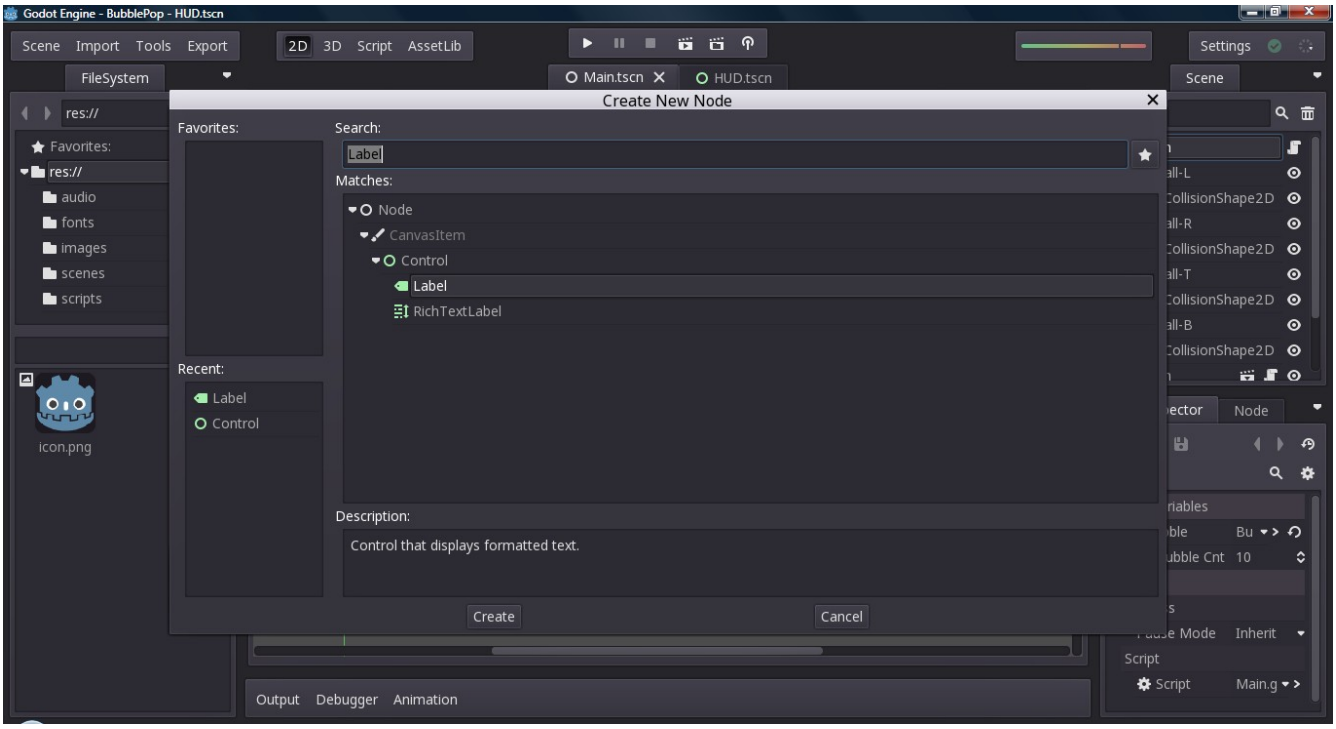


Now our labels are correct. However, we still need to add our HUD to the main scene. Open the “Main” scene:

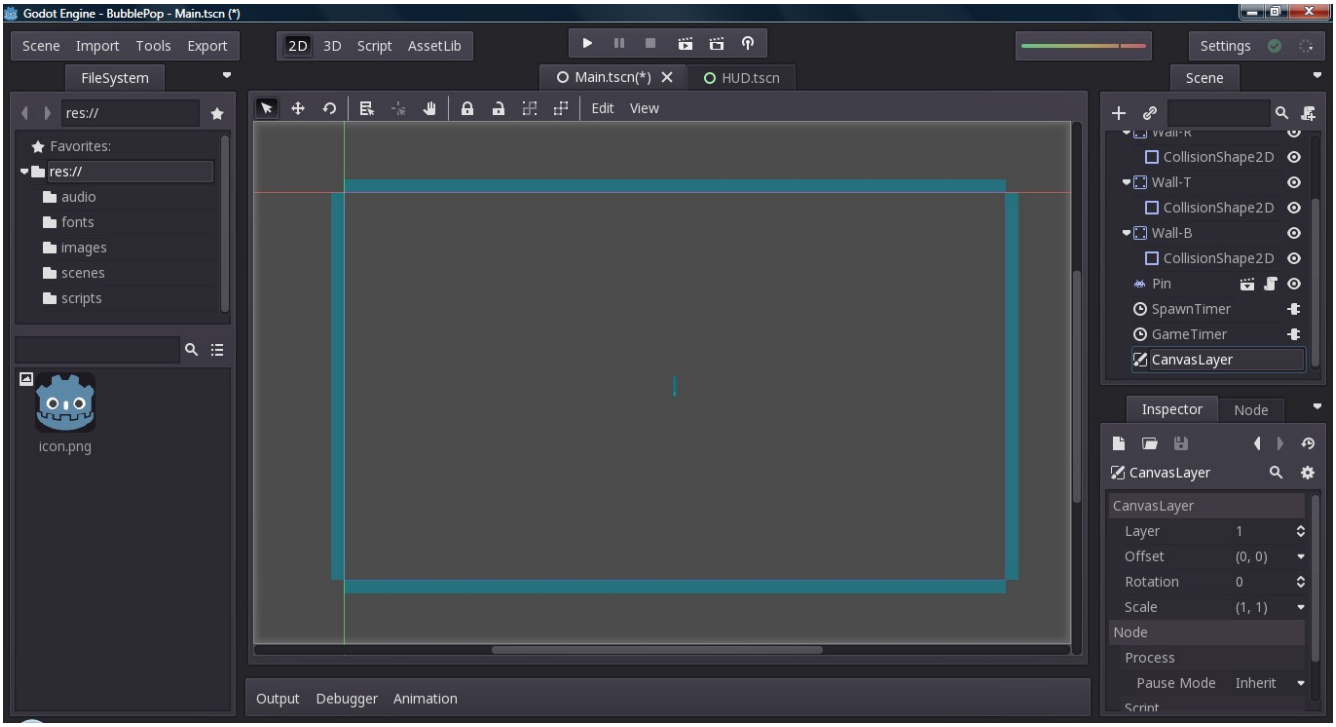


Now you may be thinking “I just need to add an instance of the HUD scene to the Main scene.” However, if we add the HUD as a child of the “Main” node, our bubbles will be on top of the HUD regardless of the order of the child nodes. Why is that? Because our bubble instances are always created after the static scene

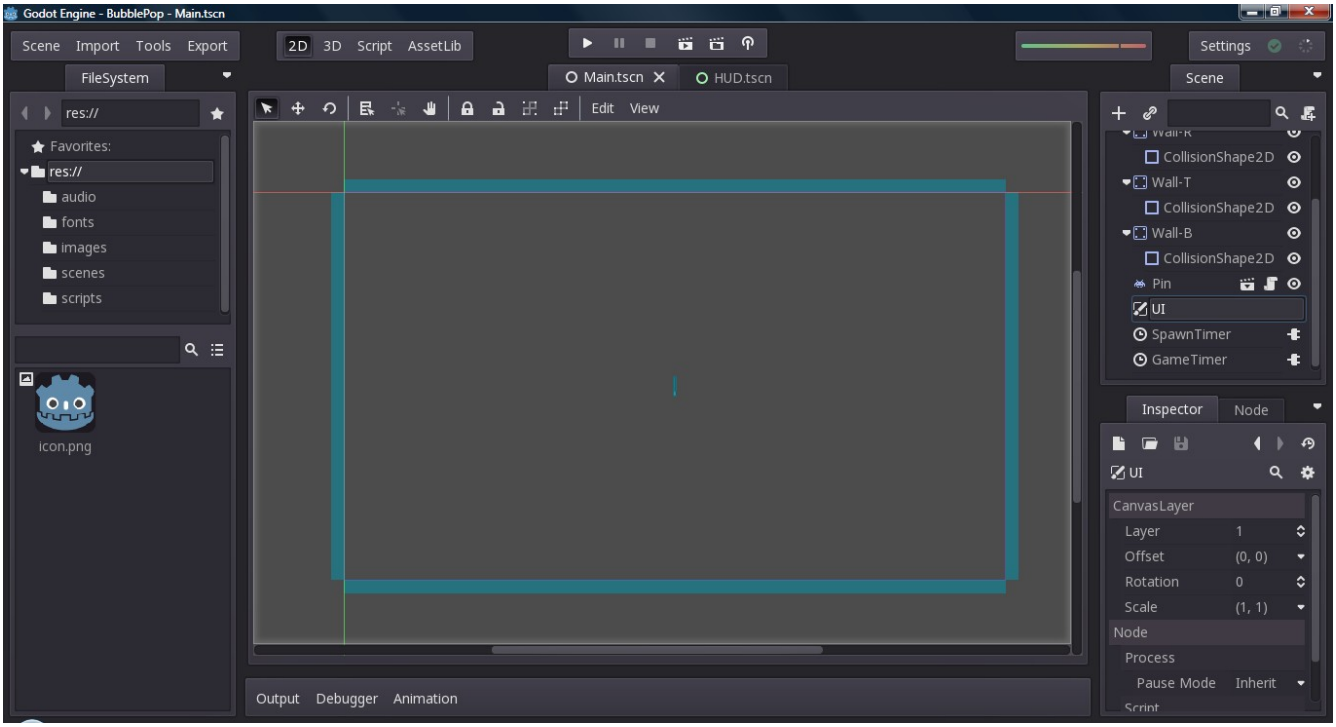
elements. Which means that they will always be on top. How can we get around this? There is actually a special type of node for this very purpose. Right-click the “Main” node and choose “Add Child Node”:



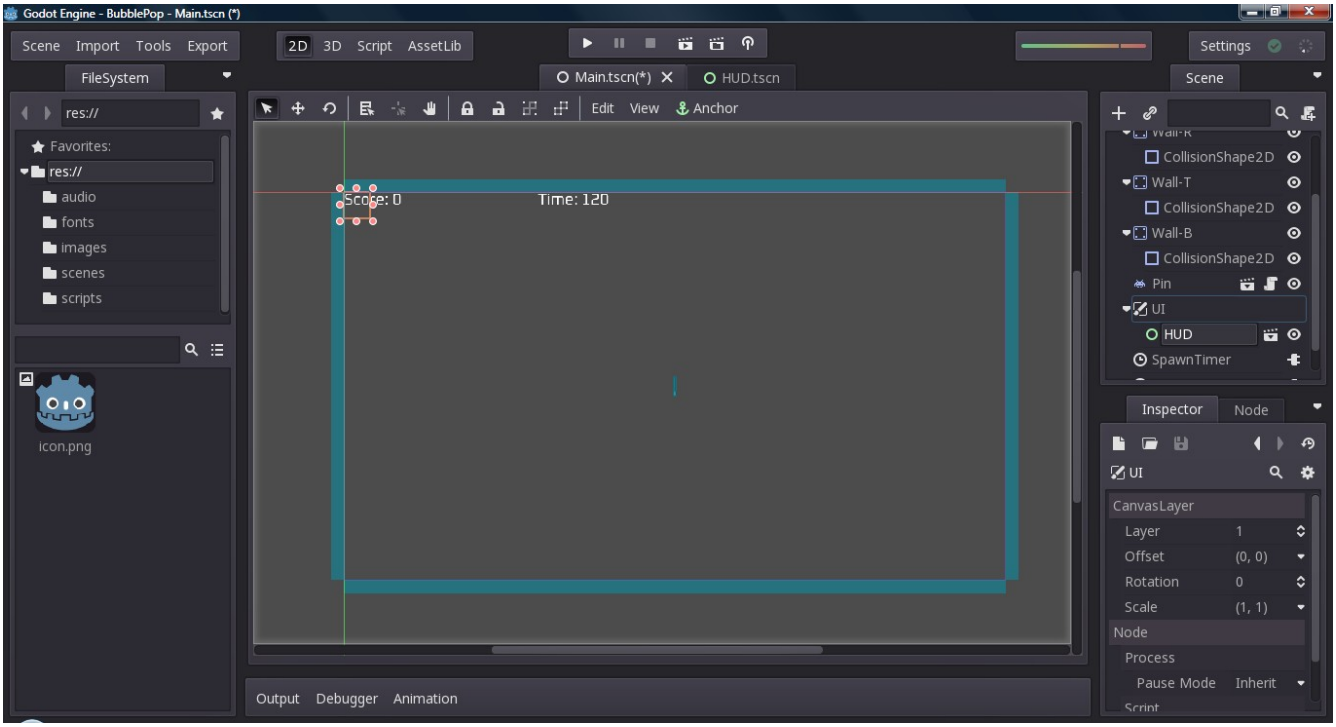
Now select the CanvasLayer node and click “Create”:



Change the name of the new node to “UI” and move it above the 2 timers:



A CanvasLayer node allows us to ensure that a group of nodes will render on top of the other nodes. It is also possible to have multiple CanvasLayer nodes. Now let's add an instance of our HUD to the "UI" node:



If we run our game now, the HUD will display over top of the bubbles:

We still need to write code to update the HUD when the score and time variables change though. First we will switch back to our "HUD" scene and attach a new script to the root node. Then we will add 2 new functions to it:

```
func set_score(score):
	#Update the score label
	get_node("ScoreLabel").set_text("Score: " + str(score))


func set_time(time):
	#Update the time label
	get_node("TimeLabel").set_text("Time: " + str(time))
```

These new functions will provide an easy way to update the score and time labels by calling them from our main scene. The "set_text" method of a label replaces the current text of the label with the new text we give it. However, our score and time values are not text, they are numbers. Therefore we must use the "str" function to convert them from numbers to text before we can add them to the prefix for each label.

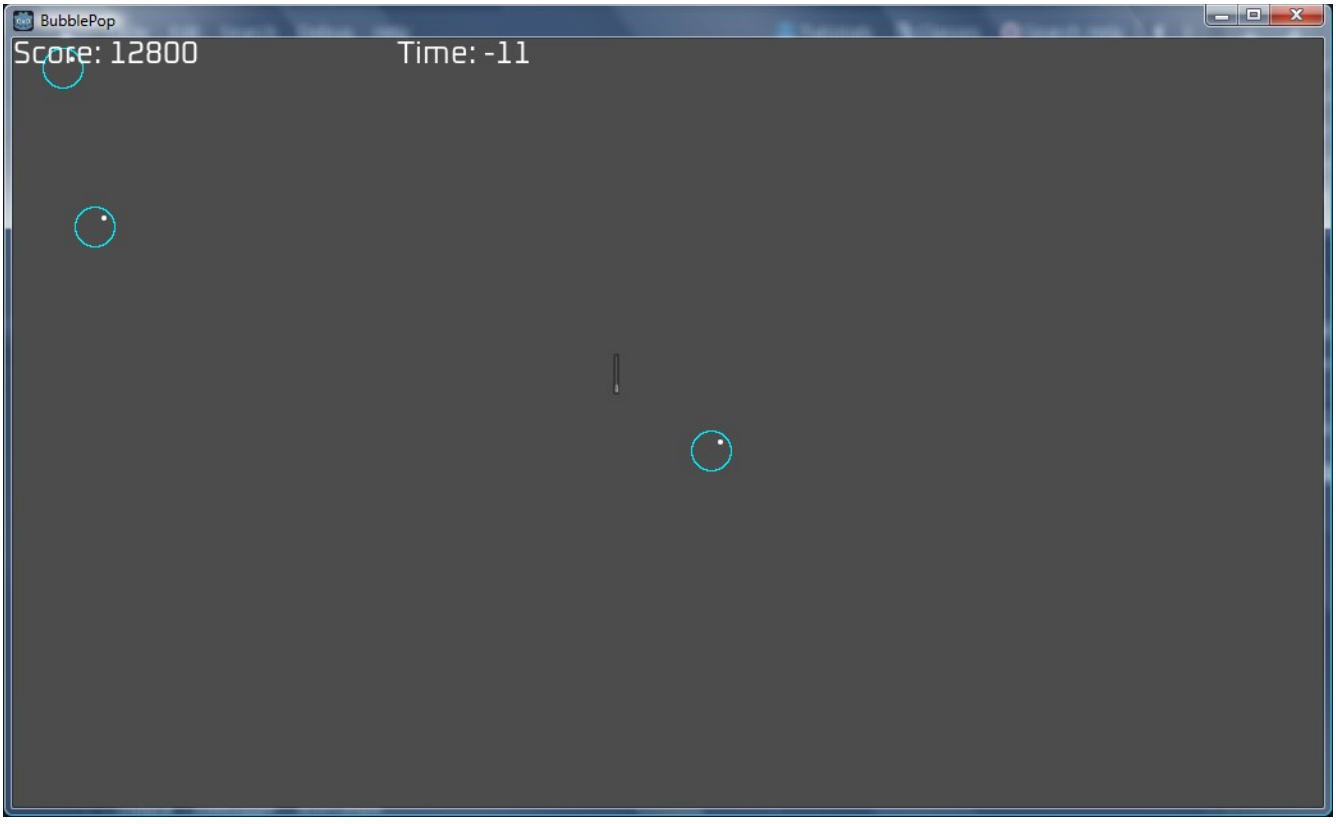
Next we will switch back to our "Main" scene and update our "_on_Bubble_popped" function:

```
func _on_Bubble_popped():
	#Decrease the bubble count by 1 and increase score
	bubble_cnt -= 1
	score += 100
	get_node("UI/HUD").set_score(score)
```

I want to point out that the node path we use this time is "UI/HUD". The reason why we cannot just use "HUD" is because our HUD node is a child of our UI node. Therefore we must give the parent node, a slash, and then the child node we want. After we have a reference to our HUD node, setting the score is easy. Now let's do the same for our "_on_GameTimer_timeout" function:

```
func _on_GameTimer_timeout():
	#Update the timer variable
	time -= 1
	get_node("UI/HUD").set_time(time)
```

If we run our game now, the score and time labels should update automatically. There is one problem though:



When the timer reaches 0, the game continues anyways and the timer goes negative. What we really want it to do is stop and display the final score. And it should also provide a way to restart the game if the player wants to try again. In the next lesson we will resolve this issue.