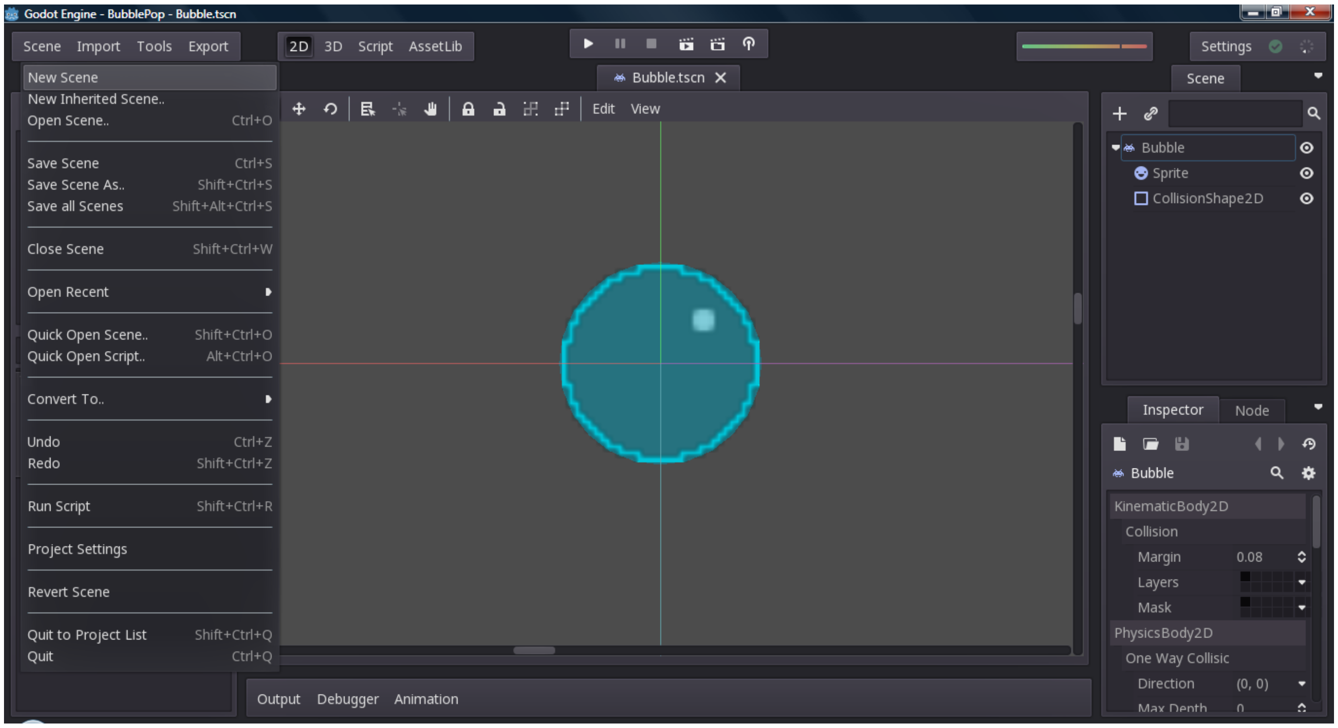
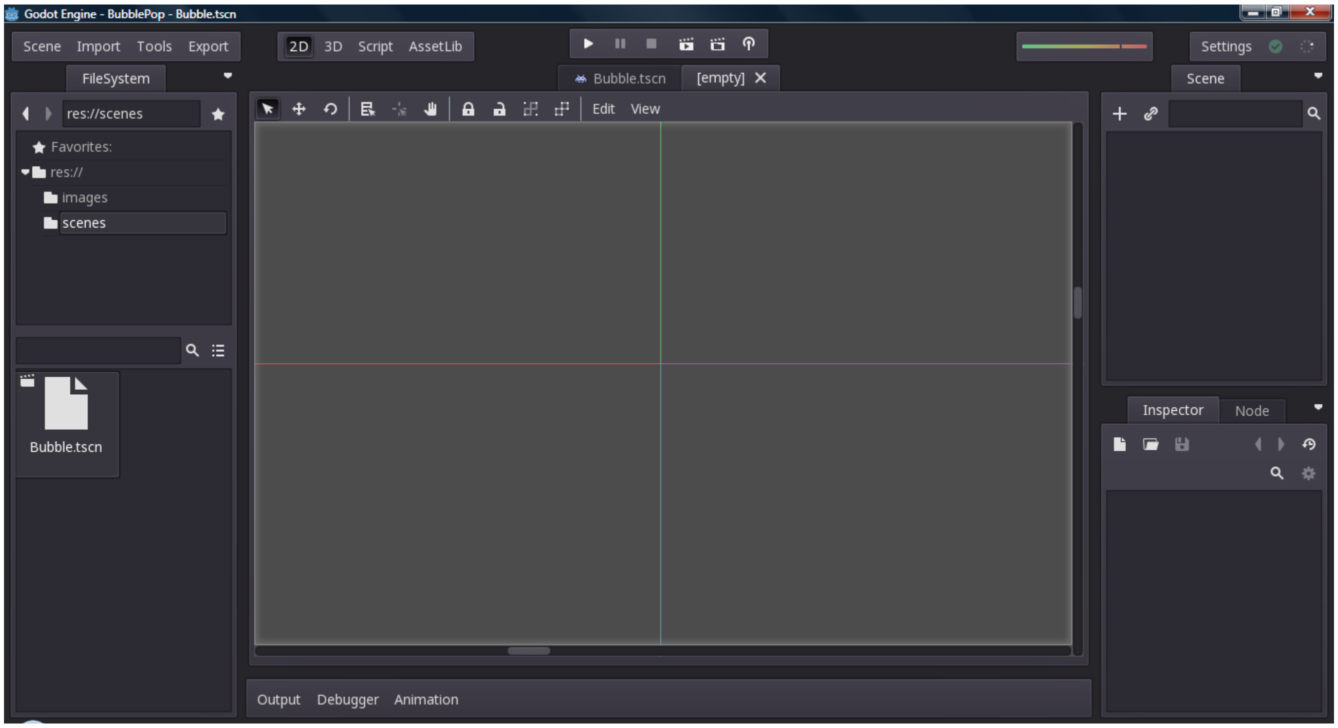
Godot 2D Game

Lesson 4: Scripting

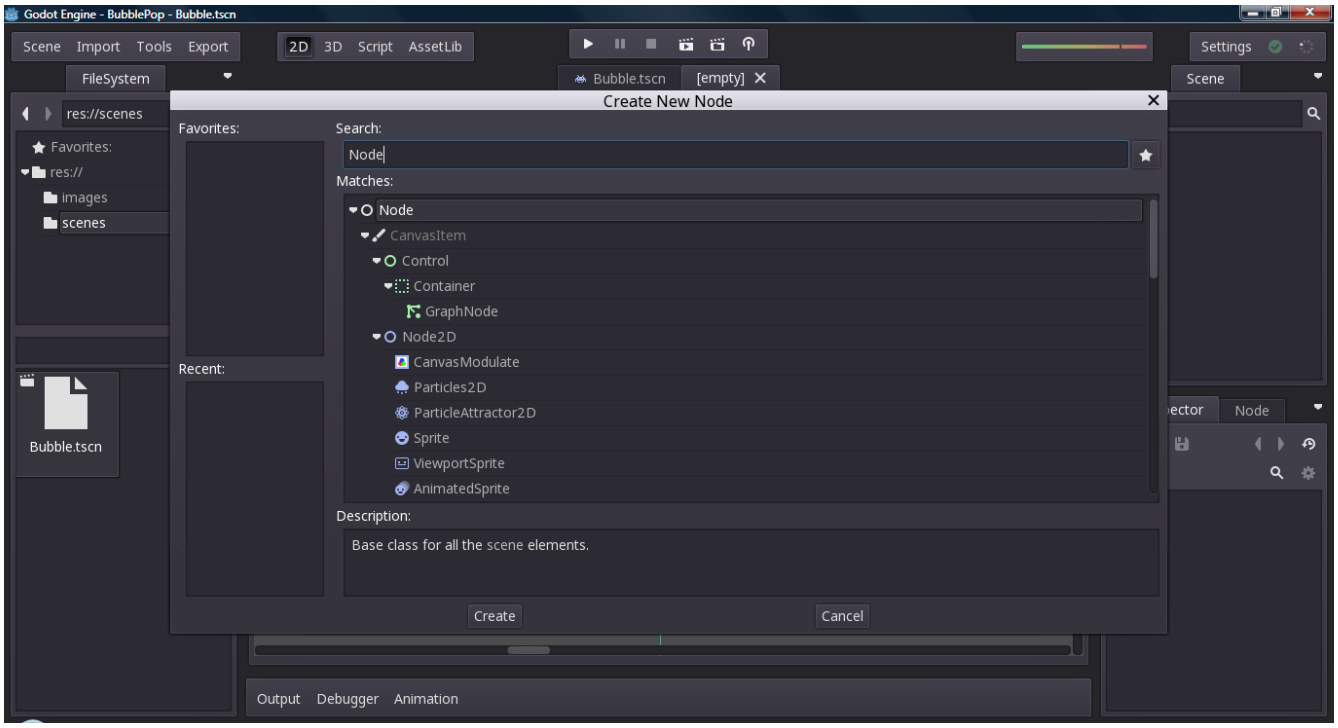
Having a bubble that just sits still and does nothing is pretty useless for a game. In order to use our new bubble we will need to learn how to use scripts. Let's start by creating a new scene called "Main" that will contain our main game logic. Click the "Scene" menu and choose "New Scene":



Notice that we now have 2 tabs on our middle workspace. Each tab contains a different scene. We can close a scene by clicking the X on its tab and we can open a scene by double-clicking it in the lower left pane after selecting our "scenes" folder in the upper left pane:



Let’s start by creating our root node for this scene. But this time, we are going to use a generic scene node:



Create New Node

Favorites:

Recent:

Search:

Node

Matches:

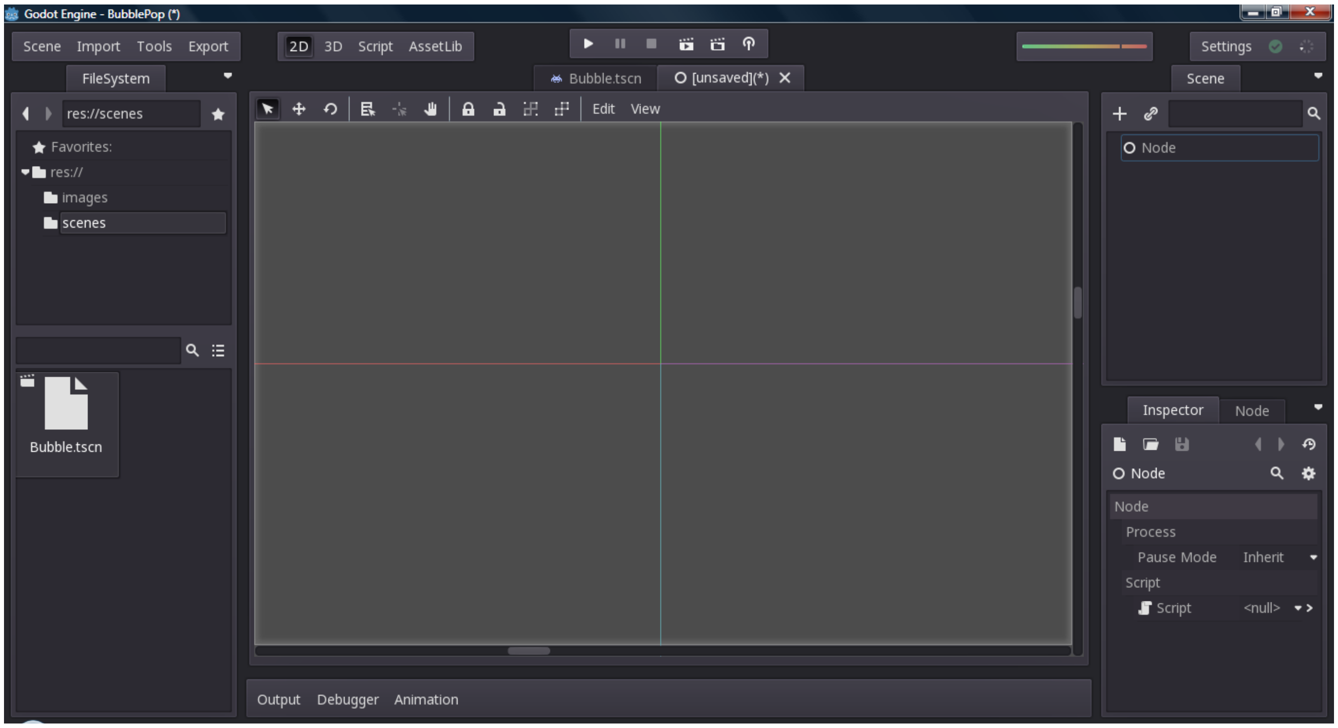
- Node
  - CanvasItem
    - Control
      - Container
        - GraphNode
    - Node2D
      - CanvasModulate
      - Particles2D
      - ParticleAttractor2D
      - Sprite
      - ViewportSprite
      - AnimatedSprite

Description:

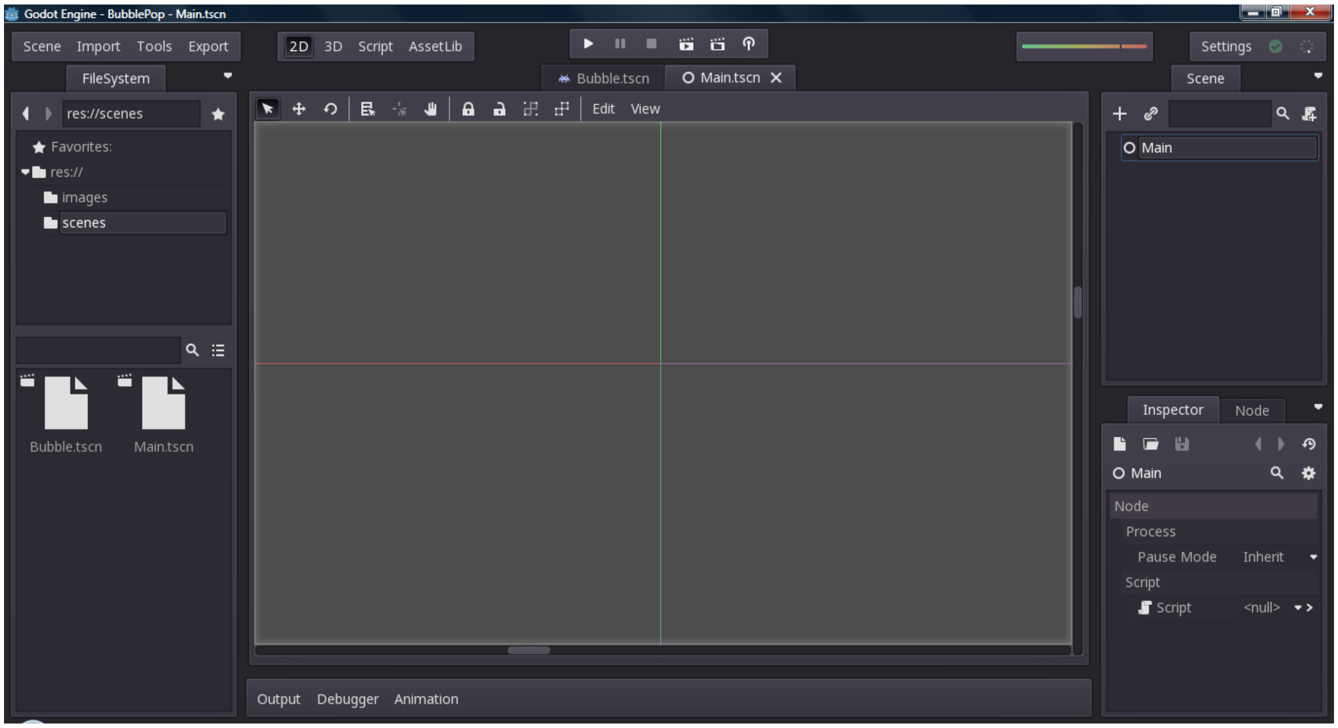
Base class for all the scene elements.

Create Cancel

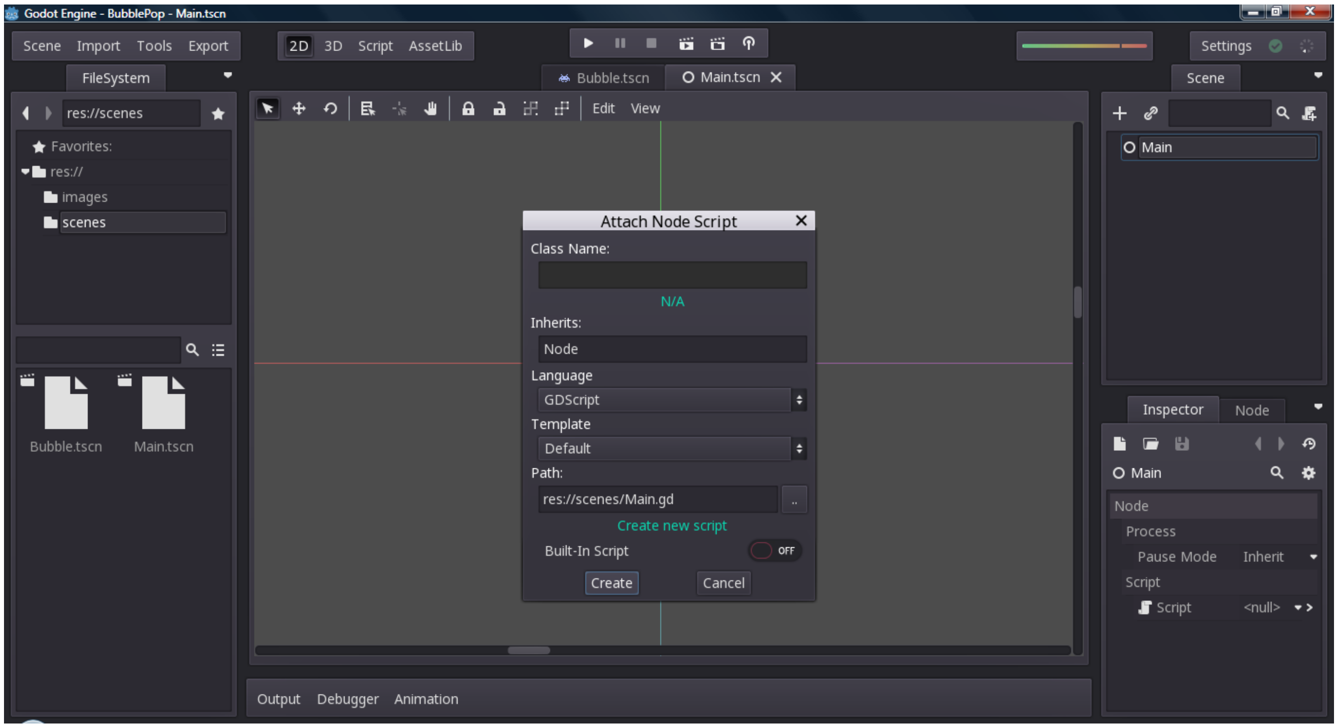
Click “Create” and our scene should look like this:



Now I bet you are wondering what is so great about a generic scene node. After all, it doesn't do much other than acting as a container for other nodes. The reason why I chose a generic scene node for the new scene is because this will be the scene that simply provides the logic for our game. And all the game logic comes from a script rather than a specialized node. Before we continue, let's rename our root node to "Main" and save it in our scenes folder like before:



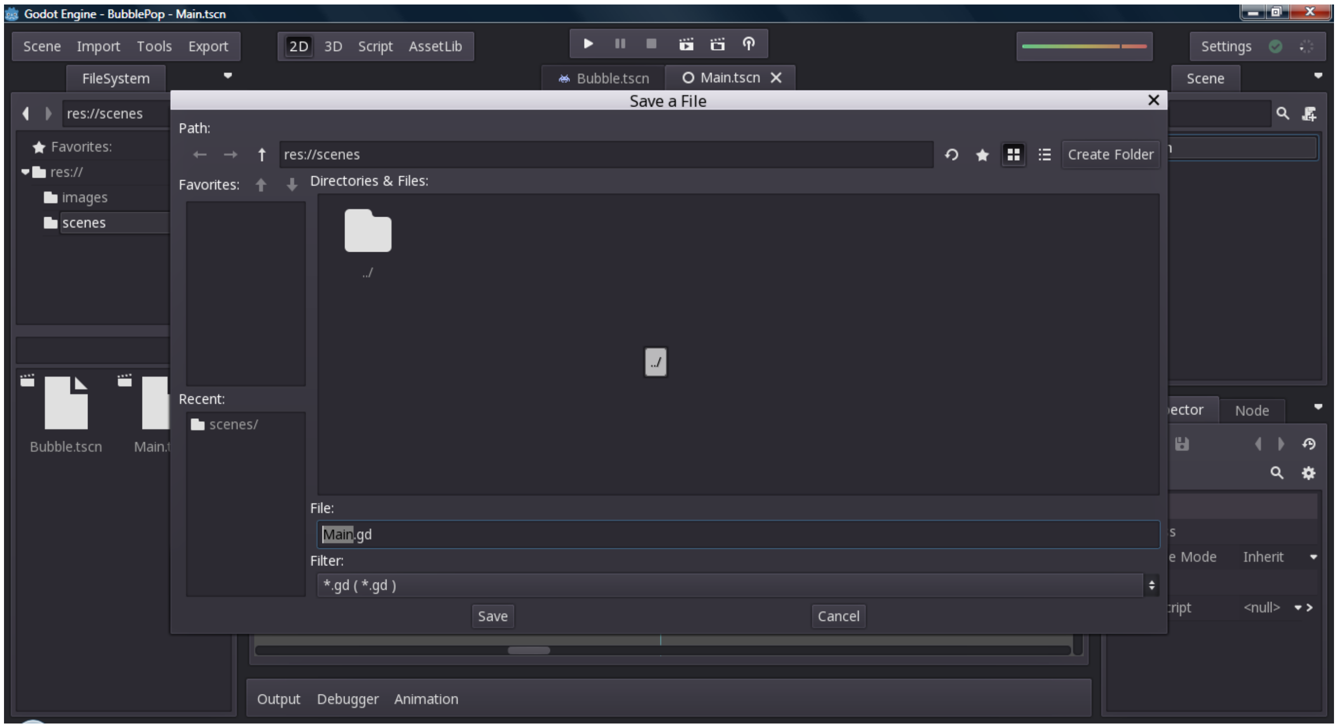
Now it is time to start adding our game logic. Right-click the "Main" node and choose "Attach Script" to open the attach node script dialog:



Attach Node Script
Class Name:
N/A
Inherits: Node
Language: GDScript
Template: Default
Path: res://scenes/Main.gd
Create new script
Built-In Script: OFF
Create | Cancel

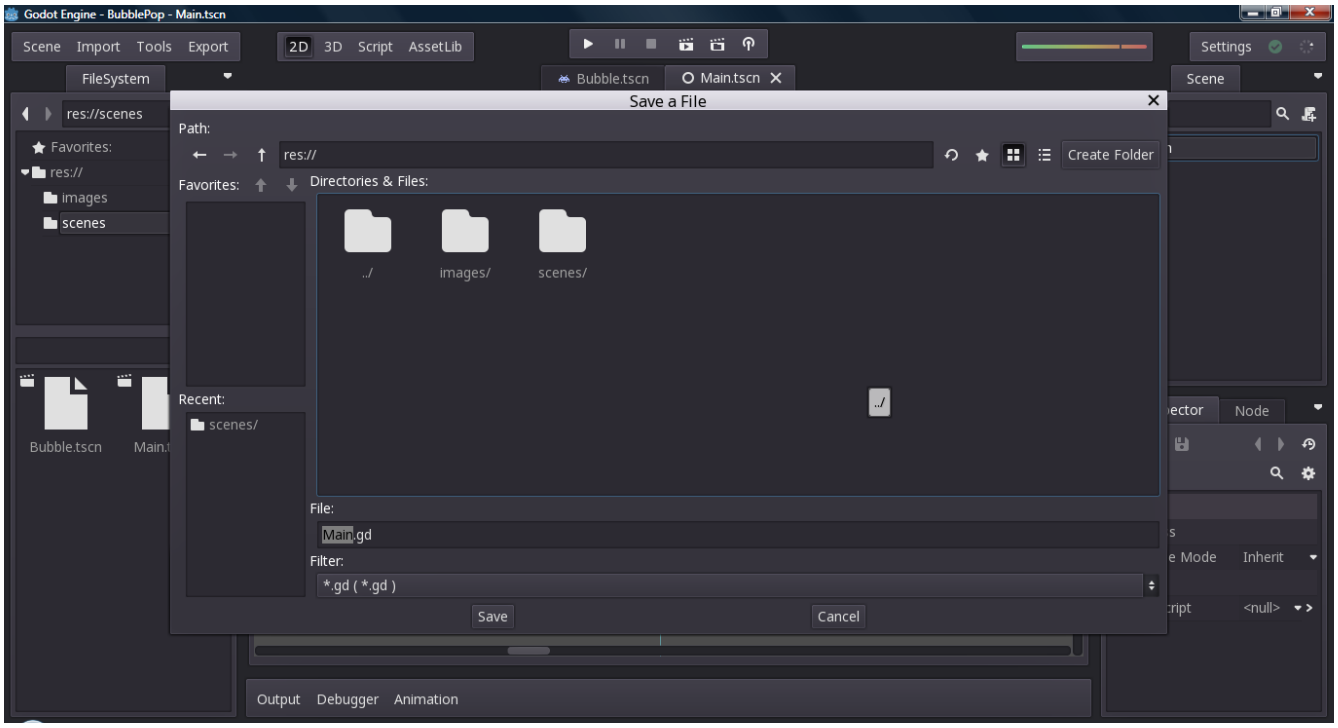
Node
Process
Pause Mode: Inherit
Script
Script: <null>

Now click the button beside the script path:



Save a File
Path: res://scenes
Create Folder
Favorites:
Directories & Files:
../
Recent:
scenes/
File: Main.gd
Filter: *.gd (*.gd)
Save | Cancel

Double-click the "../" folder to go back to the main project folder:

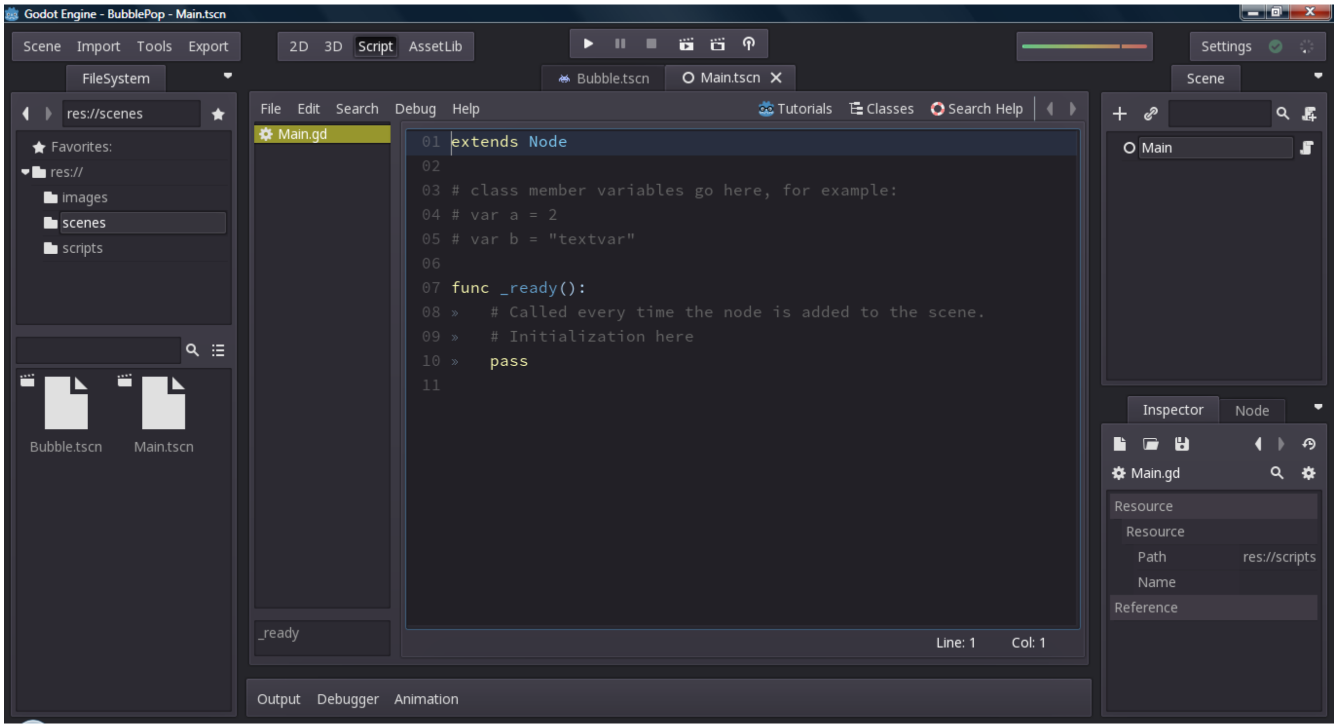


Path: res://

File: Main.gd

Filter: *.gd (*.gd)

Now create a new folder called “scripts” and click “Save”. Then click the “Create” button in the attach node script dialog and the scripting workspace will appear:



```
extends Node

# class member variables go here, for example:
# var a = 2
# var b = "textvar"

func _ready():
	# Called every time the node is added to the scene.
	# Initialization here
	pass

```

Resource
Path res://scripts
Name

This workspace is used to write the scripts that provide the logic for our game. As you can see, the list on the left of the scripting workspace contains the names of the scripts we have open. The scripting language used by Godot is called GDScript and it is similar to Python. I would also like to point out that there is a row of buttons called “2D”, “3D”, “Script”, and “AssetLib” that we can use to switch workspaces at any time. “2D” is the workspace used for editing 2D scenes, “3D” is the workspace used for editing 3D scenes, “Script” is the workspace used for editing scripts, and “AssetLib” is the workspace used for downloading and installing templates and other content made by the Godot community.

As you can see, our script already contains a few things. The first line of the script determines what sort of node the script is intended for:

```
extends Node
```

Every line that starts with a “#” is a comment. A comment is notes for the programmer that get ignored by Godot. It is always a good idea to use comments to document each part of your code:

```
# class member variables go here, for example:
# var a = 2
# var b = "textvar"
```

The comments that are automatically added remind us what each part of the generated code does. There is also an auto-generated function:

```
func _ready():
	# Called every time the node is added to the scene.
	# Initialization here
	pass
```

The “_ready” function is automatically called when the node that the script is attached to enters the scene. Notice that the contents of the “_ready” function are all indented by a fixed amount. This is required in GDScript and I should also mention that mixing spaces and tabs does not work well. The “pass” keyword is required in a function when the function is empty or only contains comments. It is essentially the “do nothing” instruction.

Now let’s discuss how our game will work. For this game, we will randomly generate up to 10 bubbles. Each bubble will have its own start position, velocity, and HP. When 2 bubbles collide, they will bounce off each other and lose HP. If a bubble’s HP reaches 0, it will pop. The bubbles will also bounce off the edges of the window. Since we intend to use randomness in our game, we will need to start by initializing the random number generator in our “_ready” function:

```
func _ready():
	#Initialize random number generator
	randomize()
```

The “randomize” function initializes the random number generator and it should always be called before generating random numbers. In the next lesson, we will start spawning bubbles.