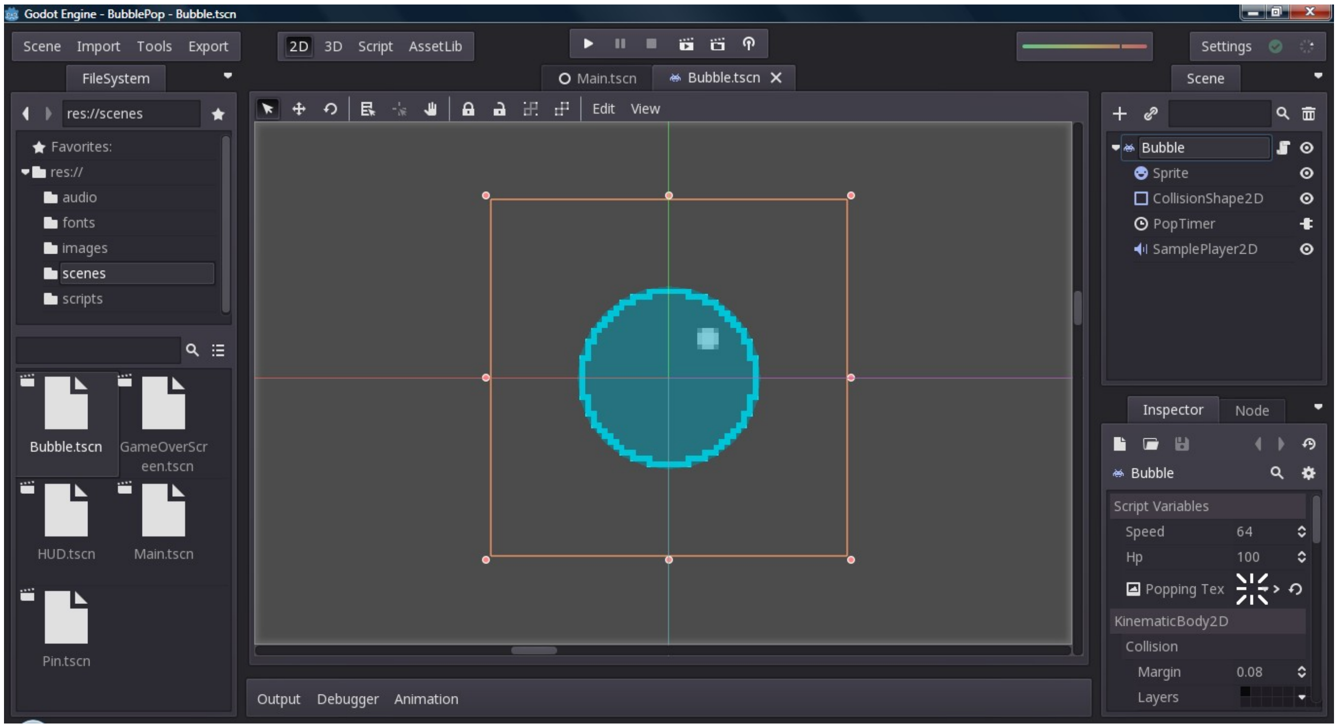
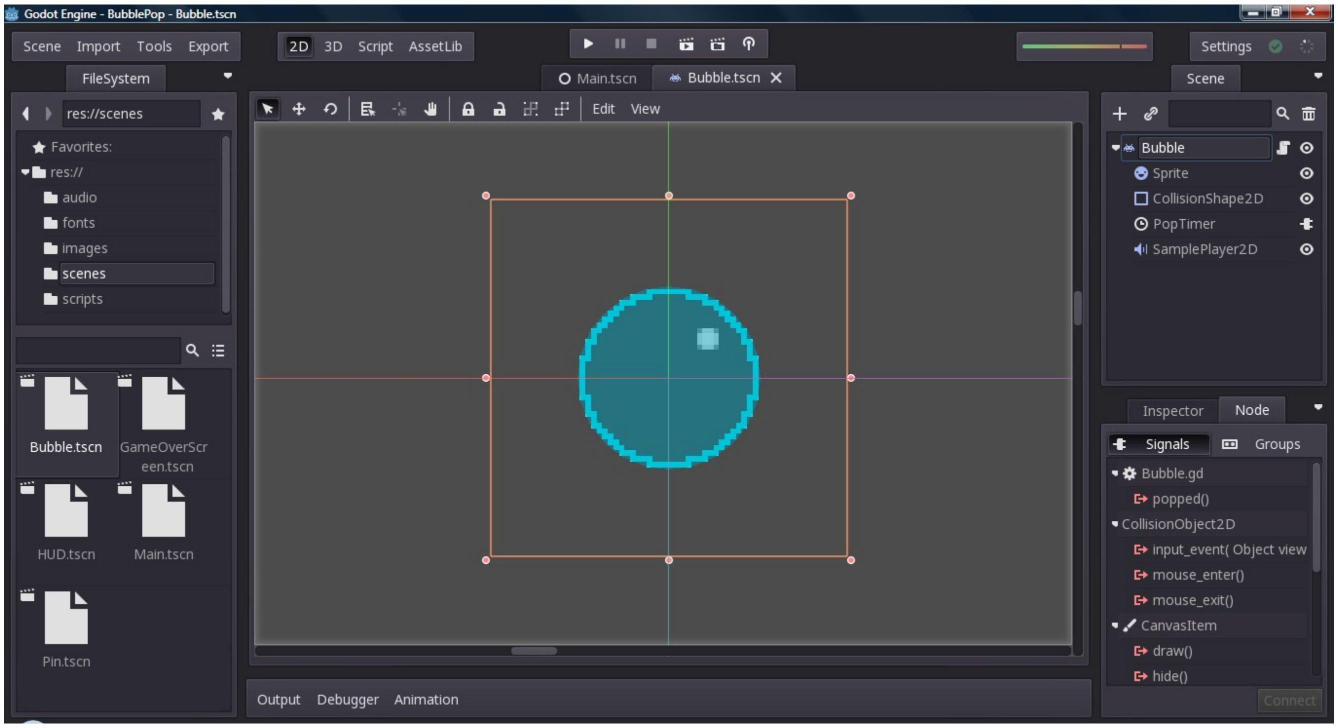
Godot 2D Game

Lesson 14: Object Groups

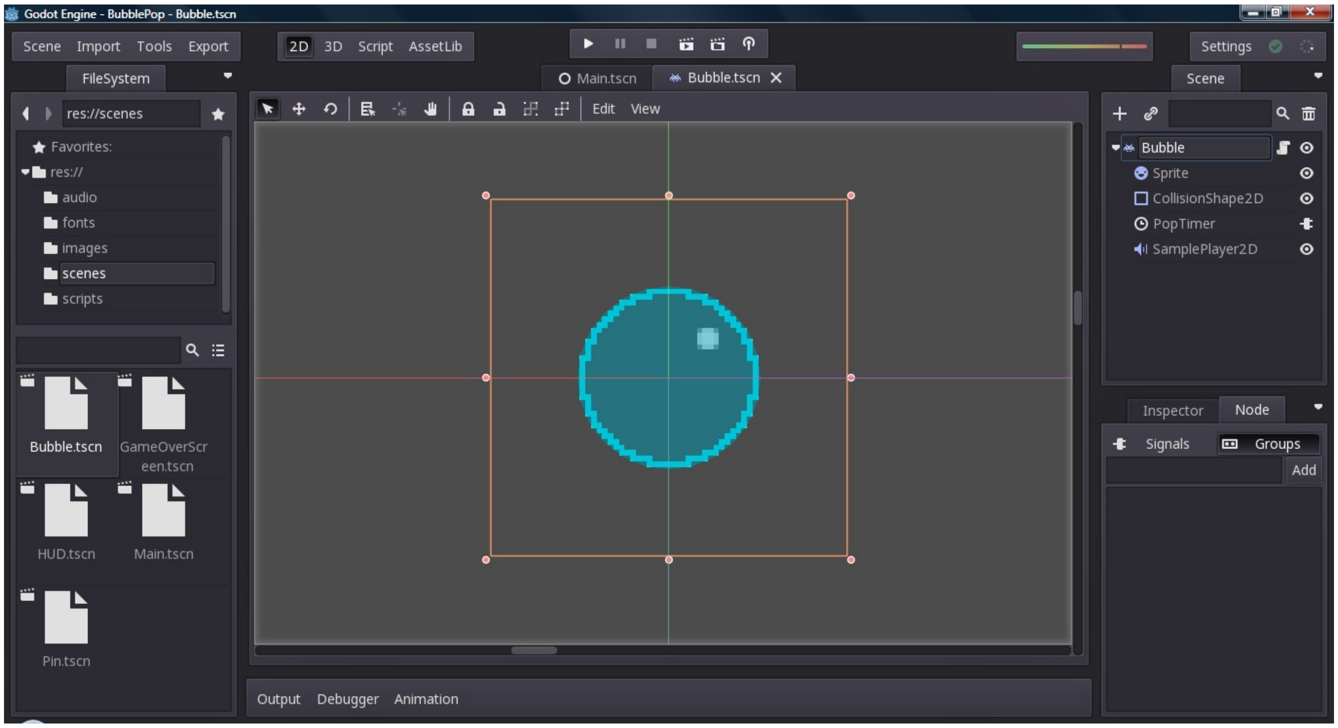
In this lesson we will tackle the task of removing all bubbles in play when the game is over. To do this, we are going to use something called object groups. First, open your "Bubble" scene:



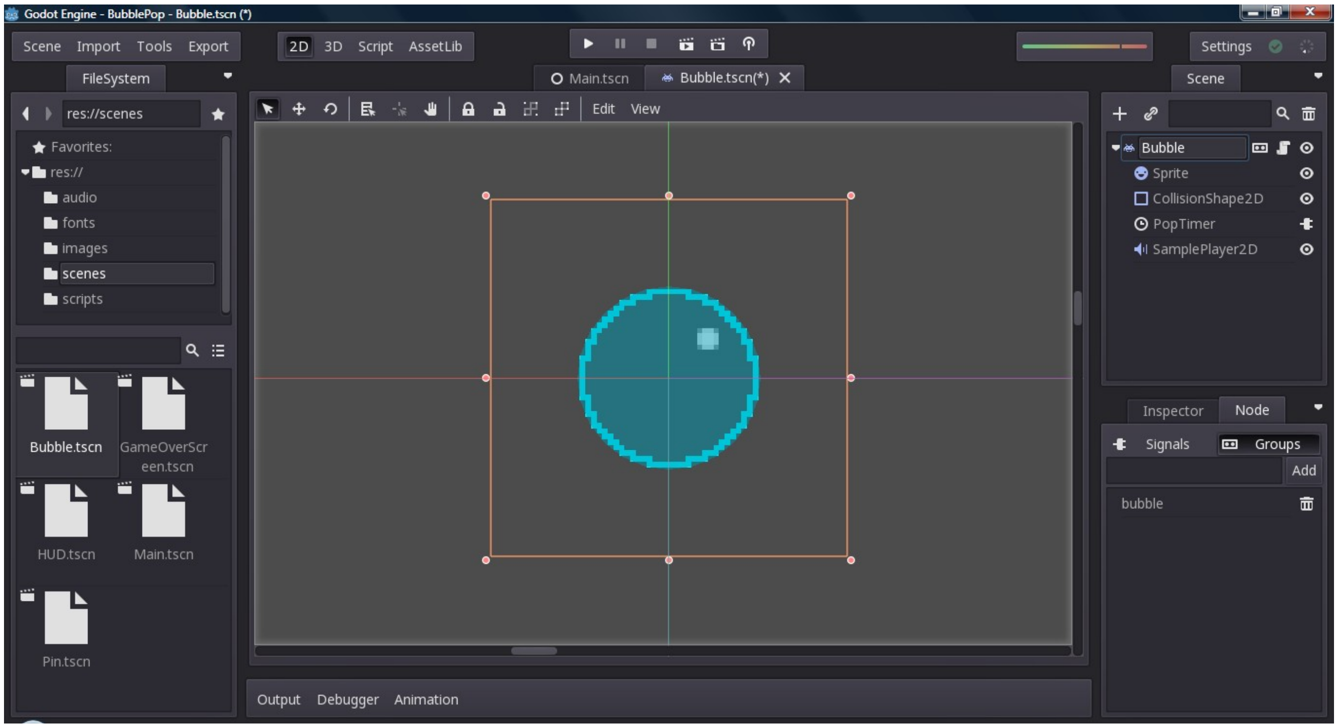
Now click the "Node" tab in the lower right pane:



Then click the "Groups" button:



Object groups provide a way to group objects in such a way that we can call a function on every object in the group at once. We can also check to see if an object is in a given group. Let's type "bubble" in the box and click "Add":



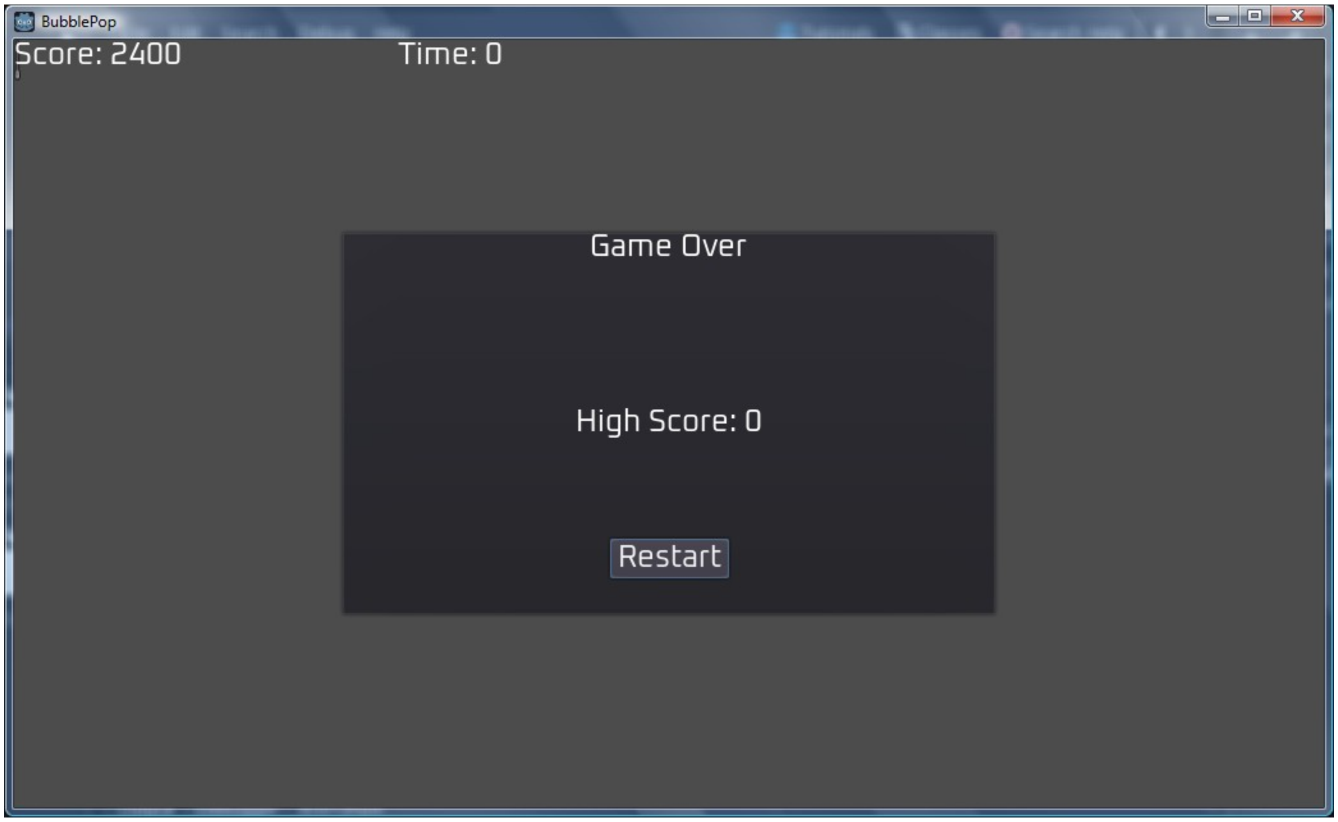
Now every bubble instance will belong to group “bubble”. This will also allow us to simplify our physics code a bit:

```
func _fixed_process(delta):
        #Update bubble position
        move(velocity * delta)

        #Update velocity
        if is_colliding():
        var collider = get_collider()
        var collider_name = collider.get_name()

        if collider_name in ["Wall-L", "Wall-R"]:
                velocity = Vector2(1, 0).reflect(velocity)

        elif collider_name in ["Wall-T", "Wall-B"]:
                velocity = Vector2(0, 1).reflect(velocity)

        elif collider.is_in_group("bubble"):
                velocity = collider.velocity.normalized().reflect(velocity)
                add_hp(-10)
```

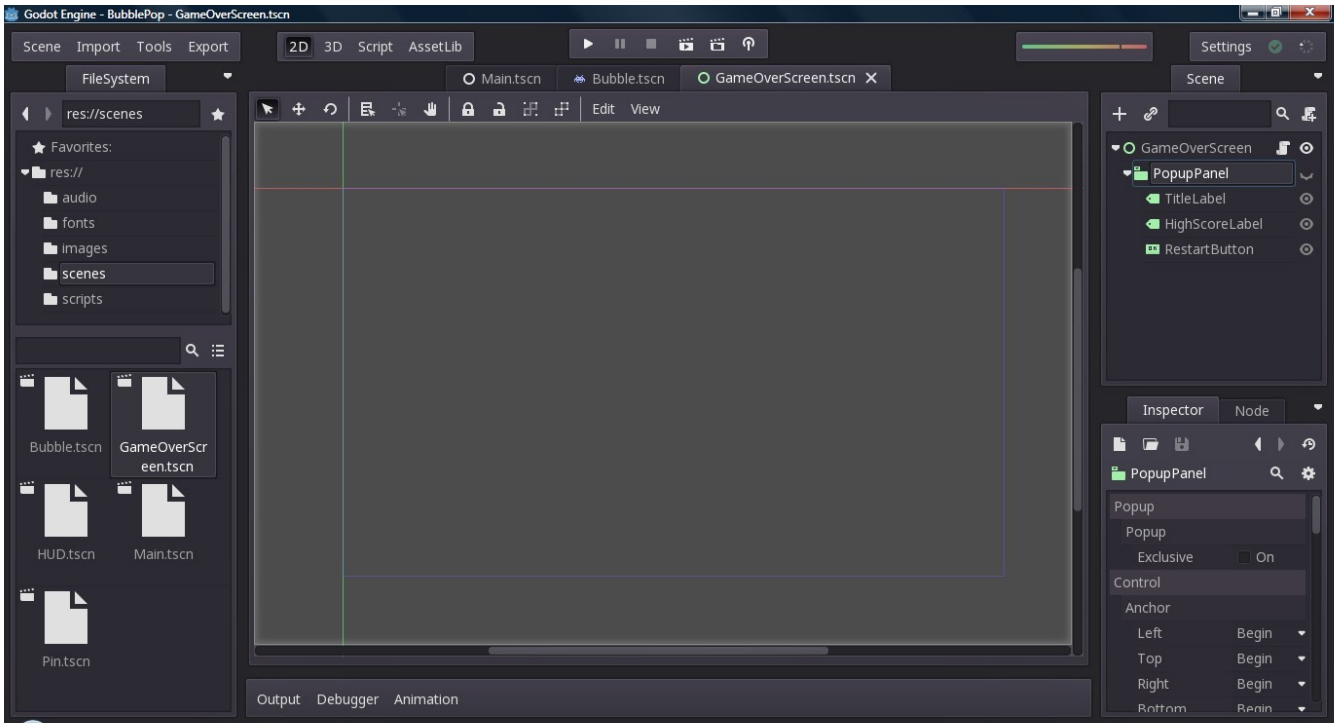
As you can see, we now have a way to test for a “bubble” by simply checking what group an object is in, rather than checking its name. It is still more convenient to check the name for the walls though. Now let’s go back to our “Main” scene and modify our “stop_game” function like so:

```
func stop_game():
	#Stop all timers, dispose of all bubbles, and show high
	#score dialog
	get_node("SpawnTimer").stop()
	get_node("GameTimer").stop()
	get_tree().call_group(get_tree().GROUP_CALL_DEFAULT,
	    "bubble", "queue_free")
	get_node("UI/GameOverScreen").show_high_score(score)
```

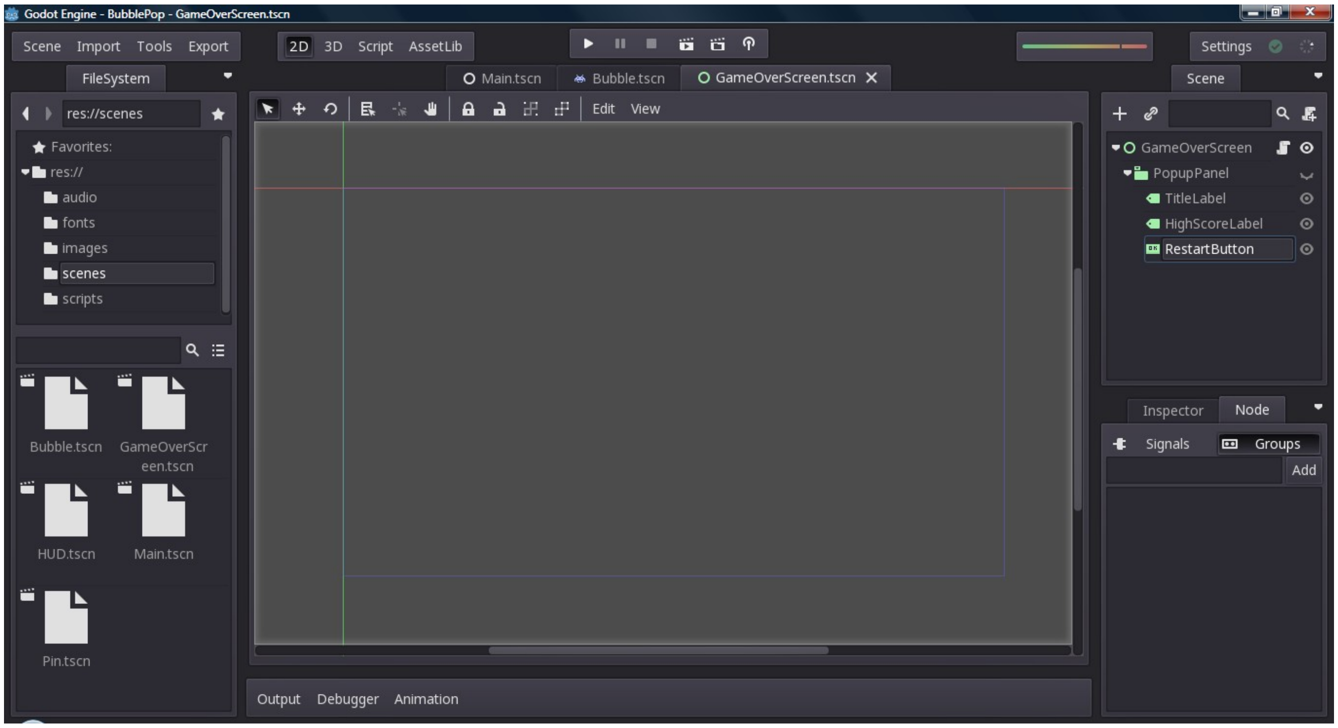
The "get_tree" function returns the tree for the current scene. If we call its "call_group" method, we can tell it to call a method on every object in a given group. Notice that we must also tell it what sort of call strategy to use too. Now if we run our game, the bubbles will all disappear when the game ends:



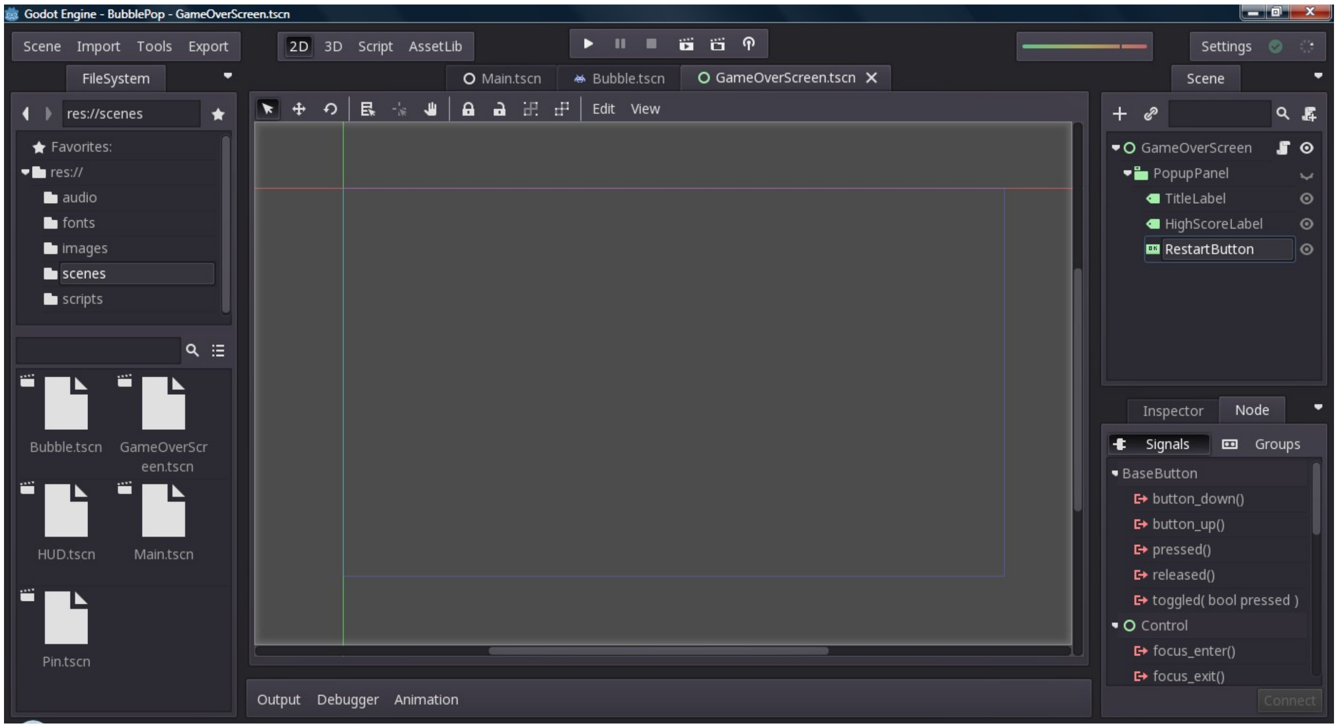
There still a few things we need to fix though. Let's start by making the PopupPanel remain open even if we click outside it. Open your "GameOverScreen" scene:



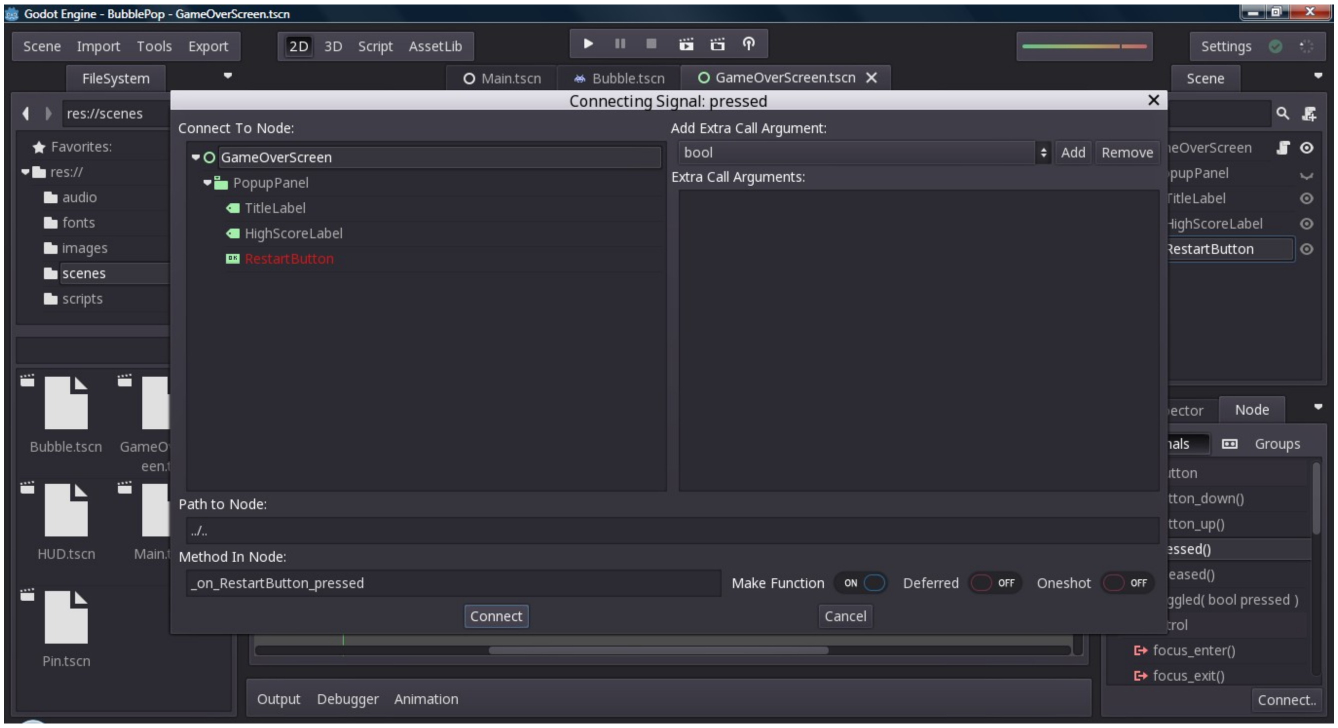
Now turn on the “Exclusive” property. This will cause the PopupPanel to not close when we click outside it. Now we need to setup our restart button so that it restarts the game when clicked. First, select the “RestartButton” node. Then click the “Node” tab in the lower right pane:



Click the “Signals” button:



Now select the “pressed” signal and click “Connect”:



Connecting Signal: pressed

Connect To Node:

- GameOverScreen
  - PopupPanel
    - TitleLabel
    - HighScoreLabel
    - RestartButton

Add Extra Call Argument: bool Add Remove

Extra Call Arguments:

Path to Node:
../..

Method In Node:
_on_RestartButton_pressed

Make Function ON Deferred OFF Oneshot OFF

Connect Cancel

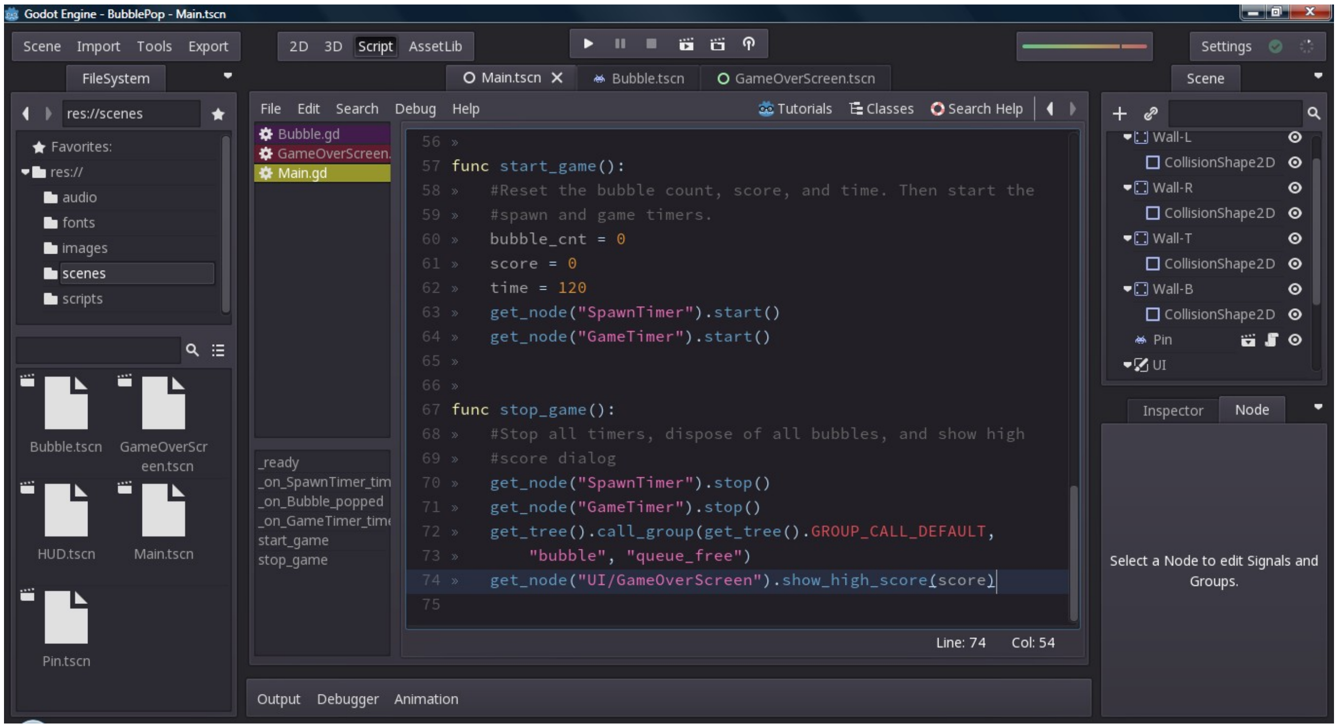
Click “Connect” again to create a new function that will be called when our button is pressed. Then we need to create a new signal below the first line of our script like this:

```
signal restart
```

We will emit this signal whenever the restart button is clicked:

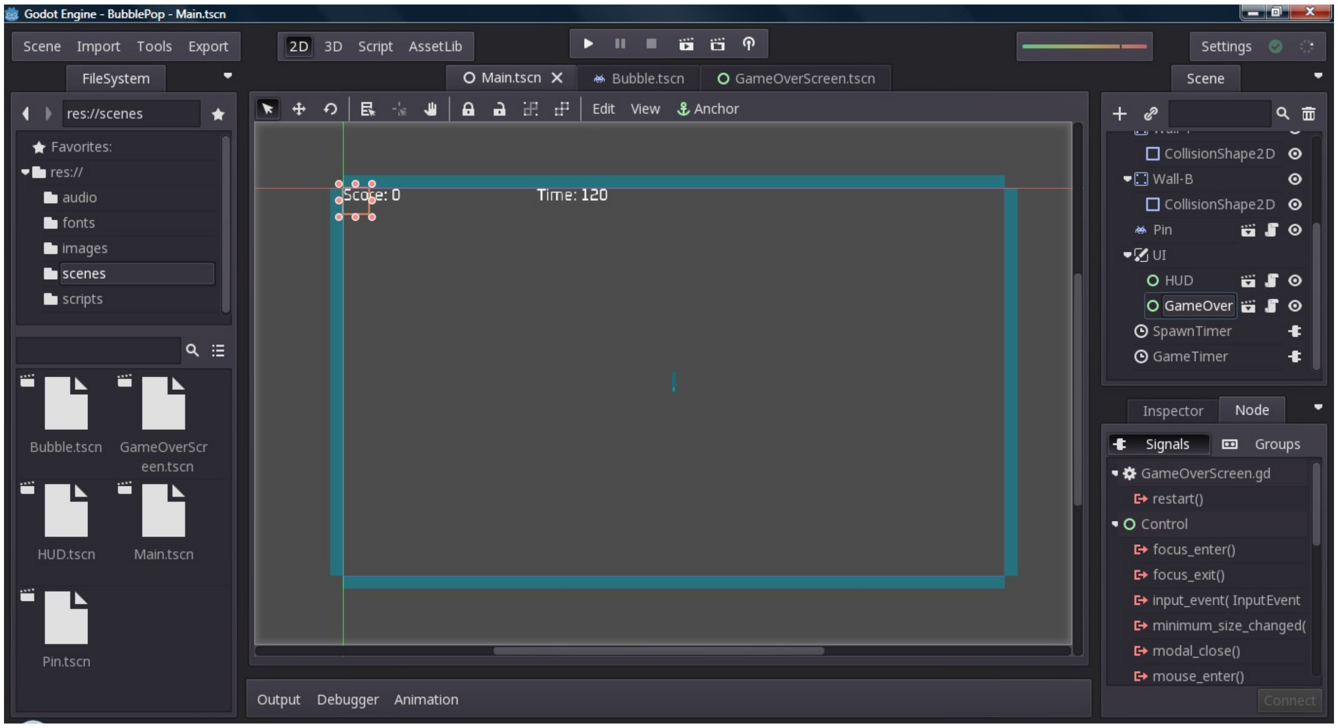
```
func _on_RestartButton_pressed():
	#Emit restart signal
	emit_signal("restart")
```

Now we need to switch back to our “Main” scene:

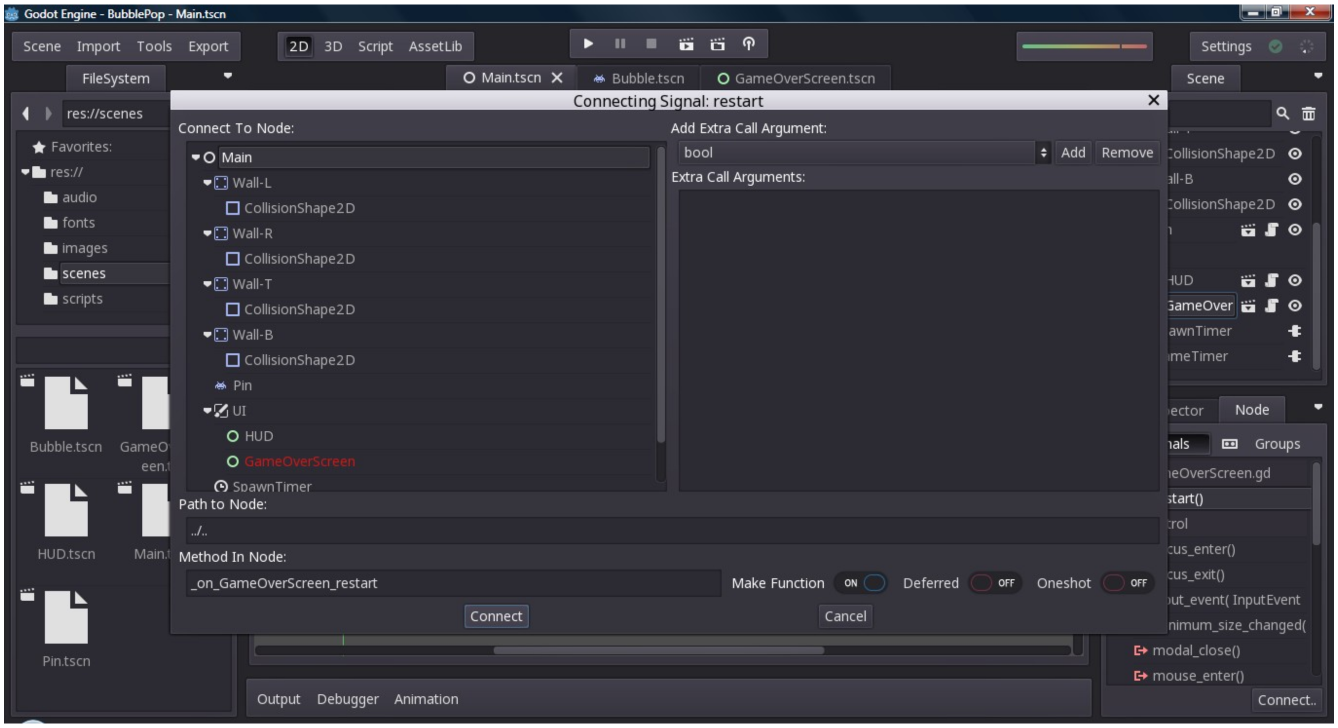


```
func start_game():
	#Reset the bubble count, score, and time. Then start the
	#spawn and game timers.
	bubble_cnt = 0
	score = 0
	time = 120
	get_node("SpawnTimer").start()
	get_node("GameTimer").start()


func stop_game():
	#Stop all timers, dispose of all bubbles, and show high
	#score dialog
	get_node("SpawnTimer").stop()
	get_node("GameTimer").stop()
	get_tree().call_group(get_tree().GROUP_CALL_DEFAULT,
		"bubble", "queue_free")
	get_node("UI/GameOverScreen").show_high_score(score)
```

Select your “GameOverScreen” node:



Select the “restart” signal from the list and click the “Connect” button:

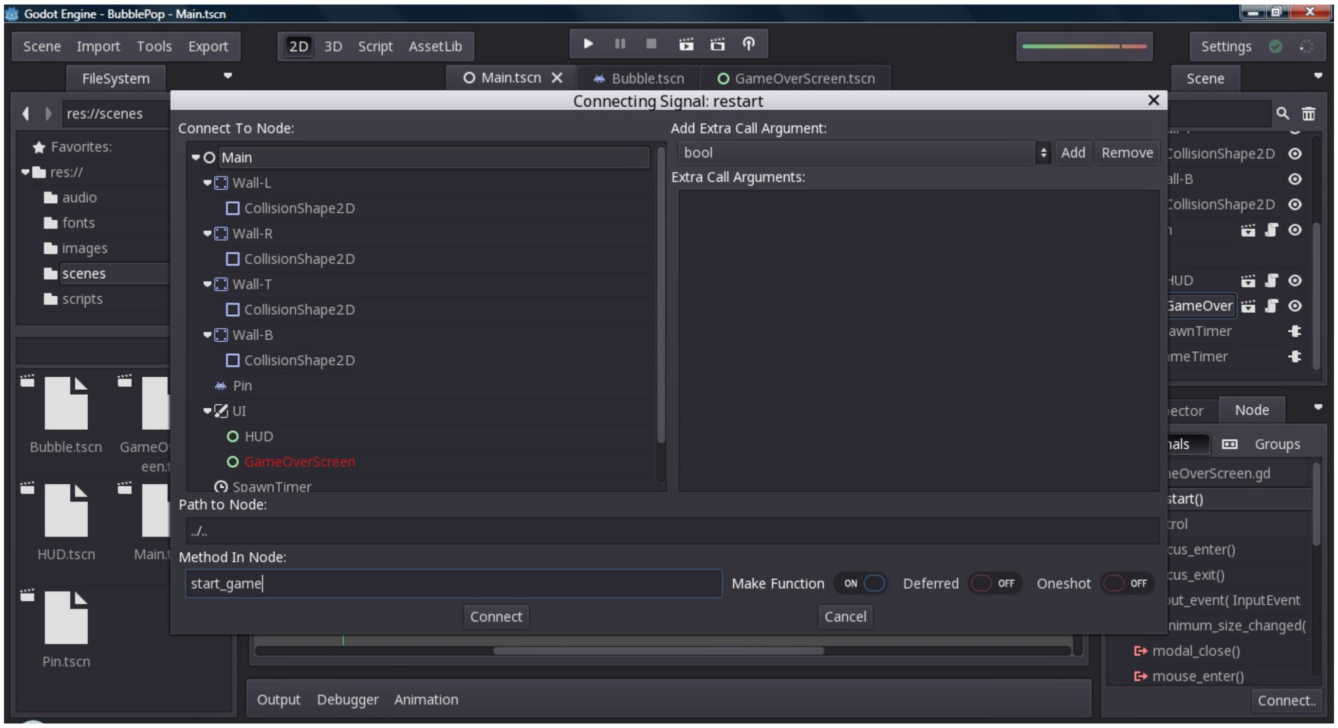


Path to Node:
../..

Method In Node:
_on_GameOverScreen_restart

Make Function ON
Deferred OFF
Oneshot OFF

This time, we are going to do something a bit different. Type “start_game” into the box at the lower left of this dialog before clicking “Connect”:



Connecting Signal: restart

Connect To Node: Main

Add Extra Call Argument: bool

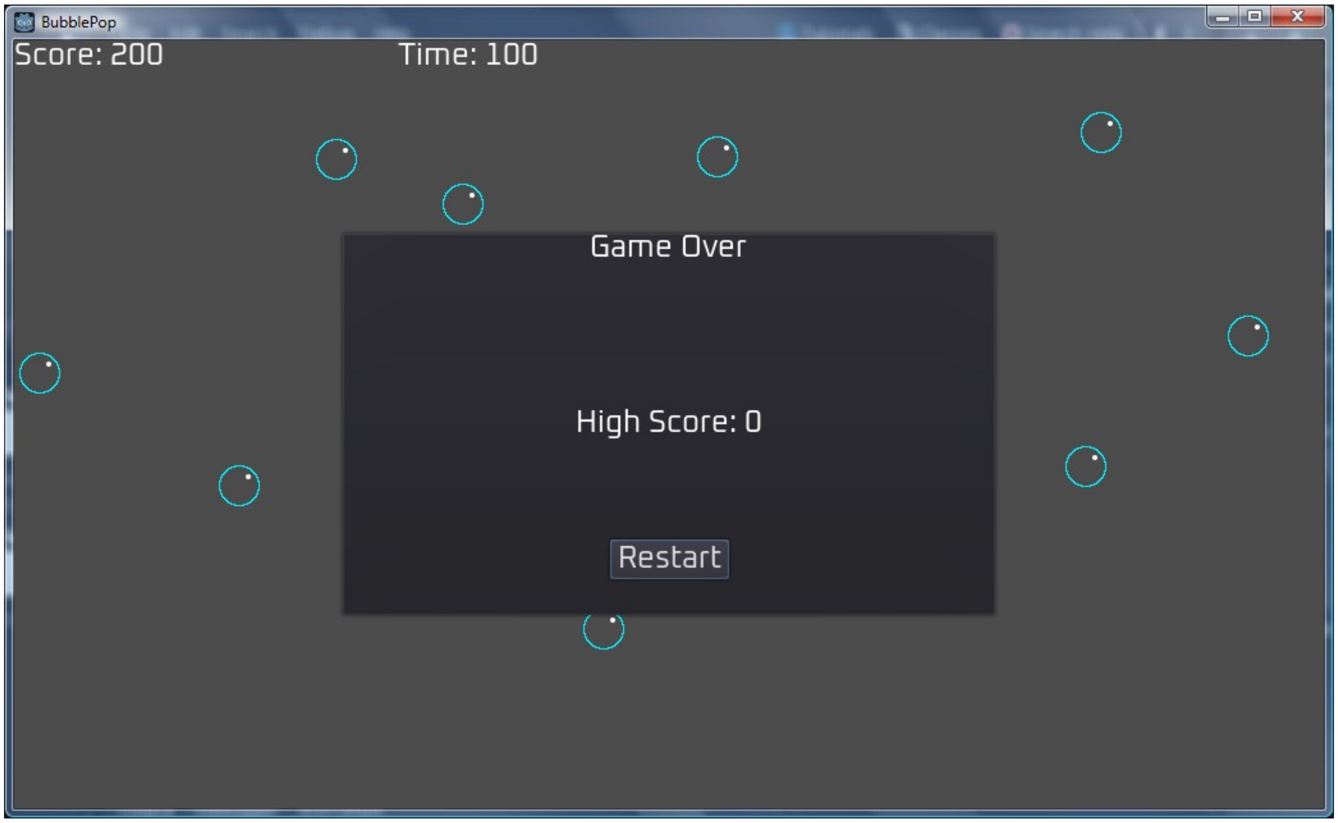
Extra Call Arguments:

Path to Node: ../..

Method In Node: start_game

Make Function: ON   Deferred: OFF   Oneshot: OFF

When you click connect, no new function will be generated. Instead, our existing "start_game" function has been attached to this signal. Now let's try running our game again:



Hmm… clicking the restart button restarts the game, but the game over screen is still visible. Let’s modify our “start_game” function to fix this:

```
func start_game():
	#Hide the high score dialog
	get_node("UI/GameOverScreen/PopupPanel").hide()

	#Reset the bubble count, score, and time. Then start the
	#spawn and game timers.
	bubble_cnt = 0
	score = 0
	time = 120
	get_node("SpawnTimer").start()
	get_node("GameTimer").start()
```

If we run our game now, it will behave as expected. In the next lesson, I will show you how to implement the high score system so that the player has something to compete against.