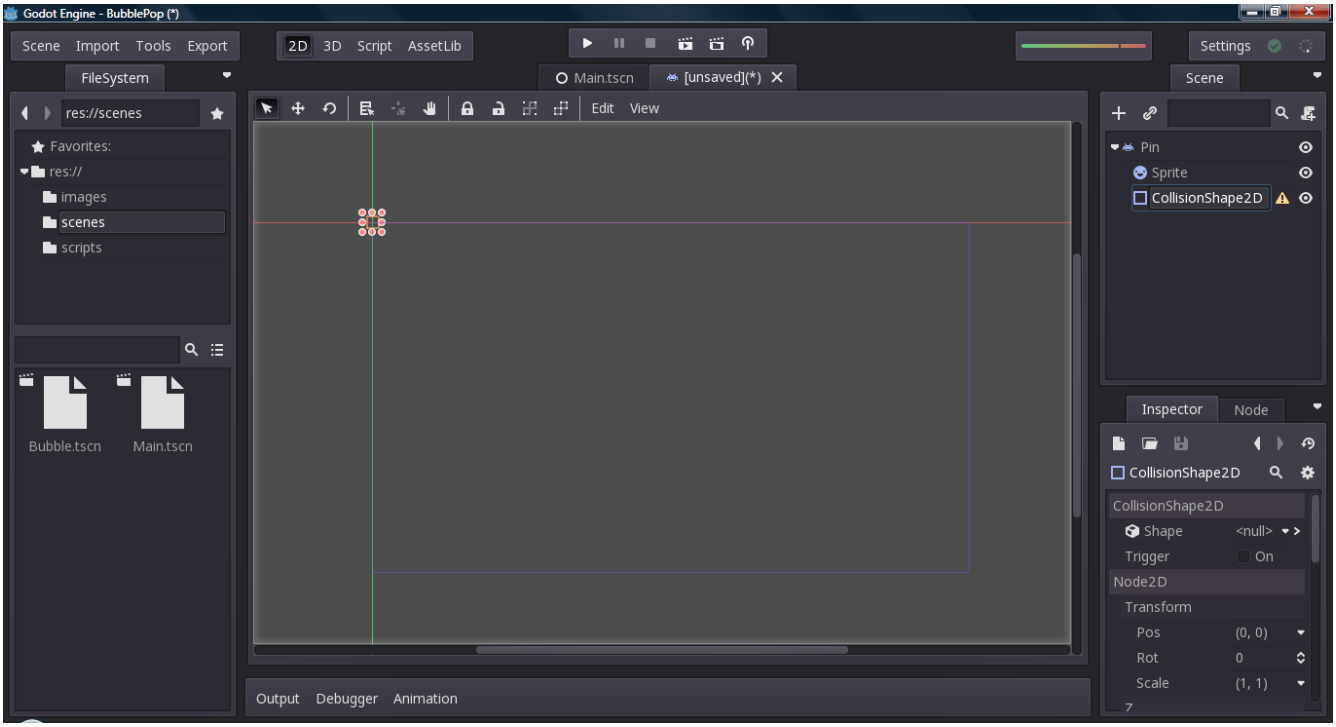
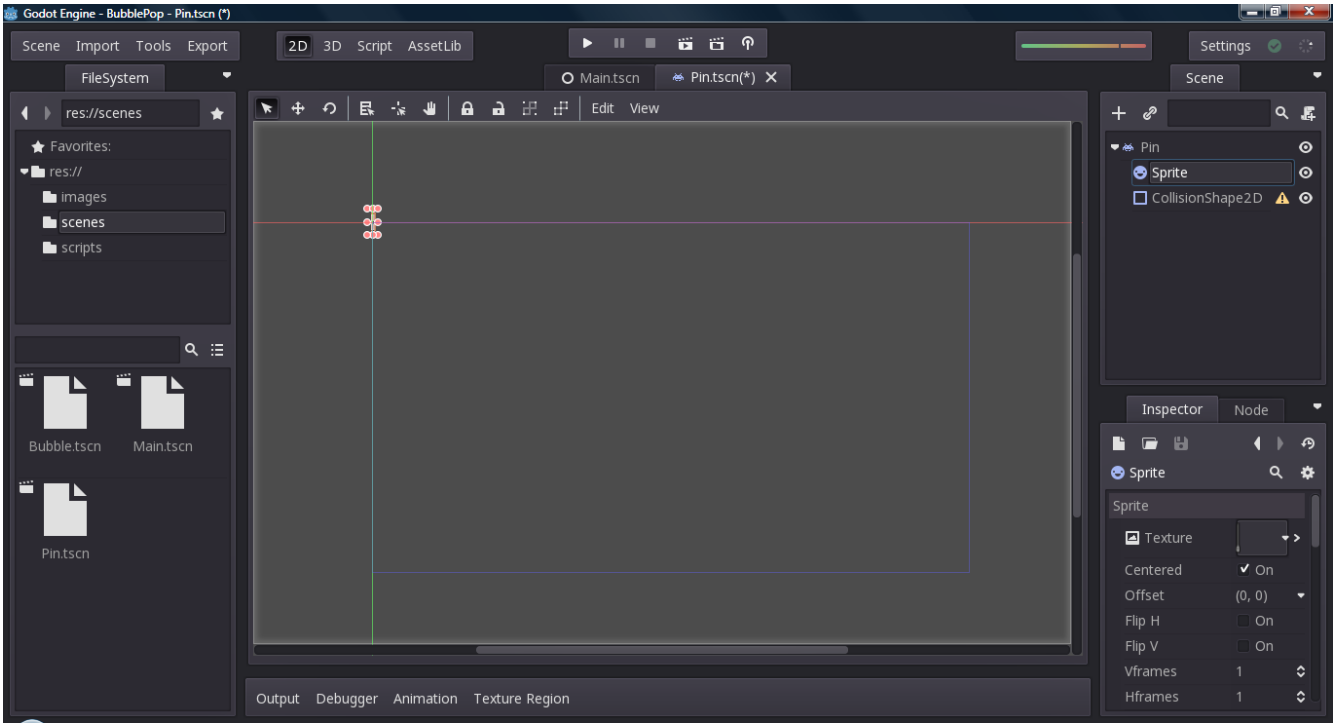
Godot 2D Game
Lesson 9: Player Input

So far, our game has not provided a way for the player to actually do anything other than watch bouncing bubbles. Such a game would obviously get boring very quickly. Let's fix that by providing a way for the player to interact with the game. In this lesson, we will add a pin that the player can use to pop the bubbles by moving the mouse.

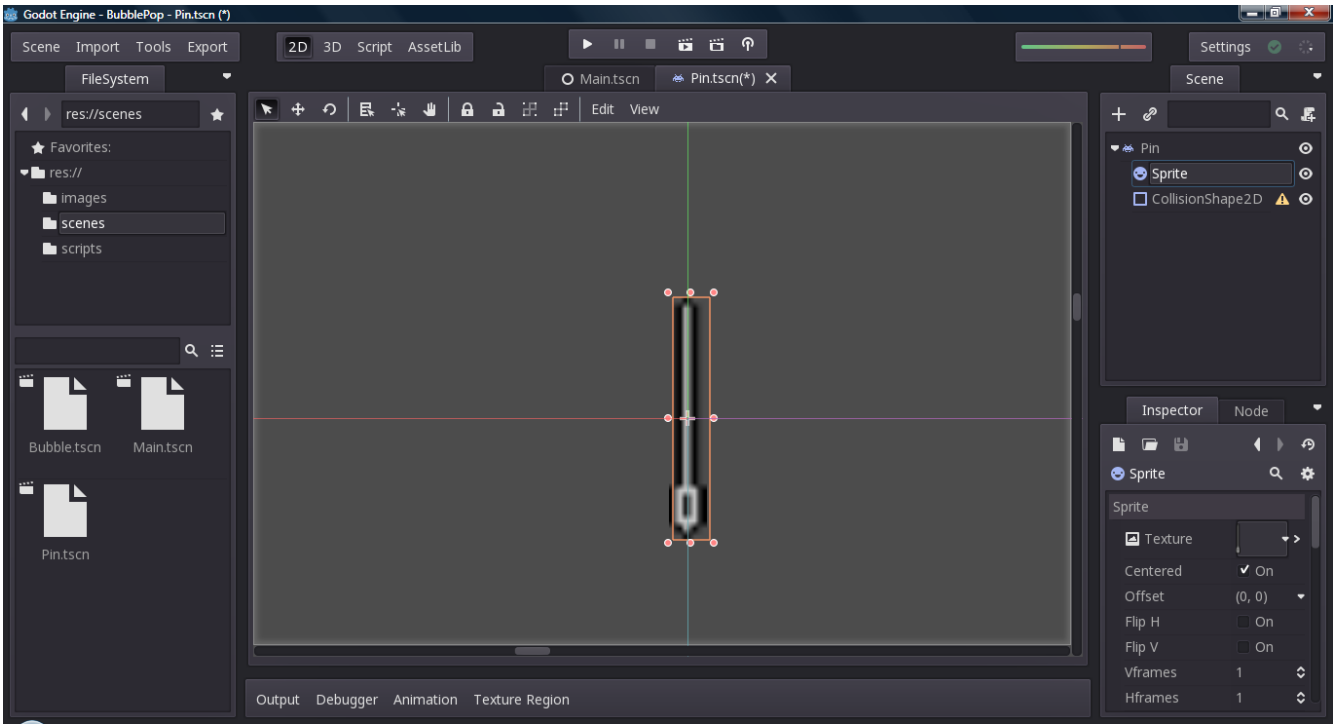
First we need to create a new scene. The root node of our new scene will be a KinematicBody2D just like our "Bubble" scene and it will also have a Sprite and CollisionShape2D as child nodes:



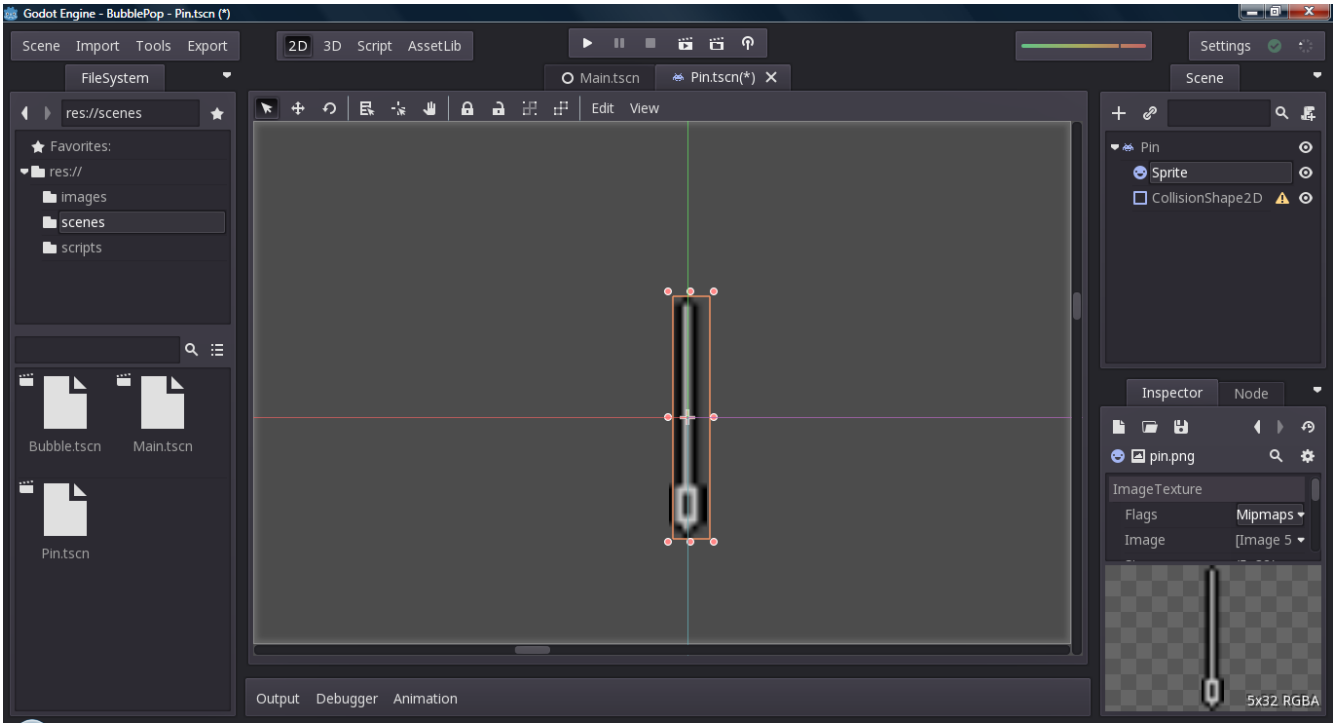
Save this new scene in your "scenes" folder before continuing. Next, we will set the texture for our sprite to "pin.png":



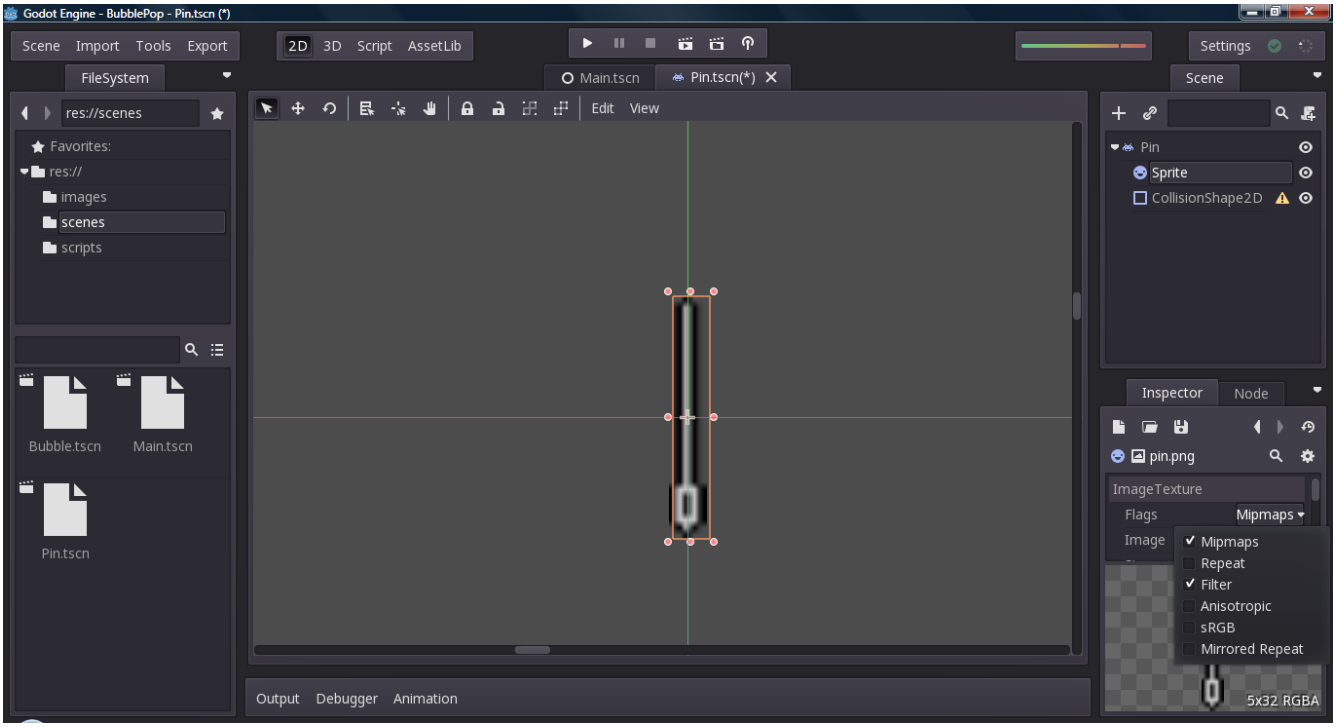
As a shortcut, we can automatically center our view on the pin by pressing the F key while the pin is selected:



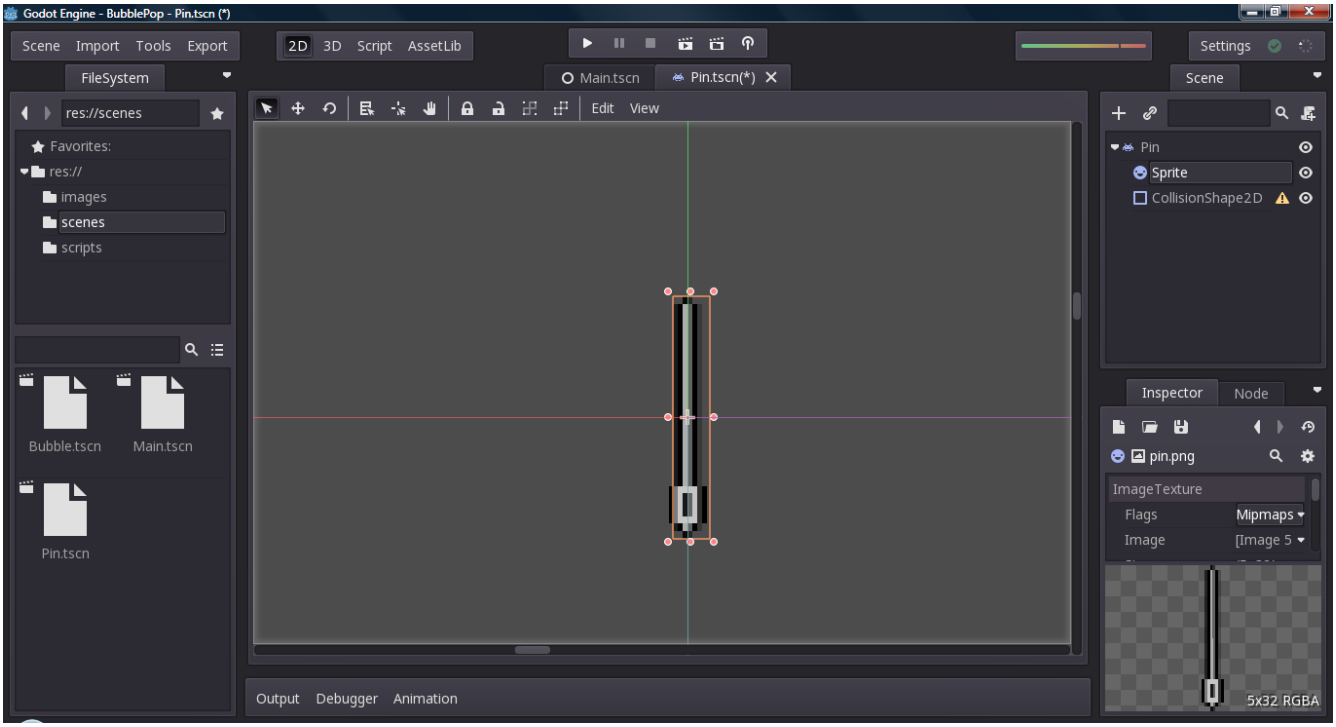
If we zoom in on the pin, we will also notice that the pin looks a bit blurry. This is a common issue with small sprites. To fix it, we need to click the arrow beside our texture:



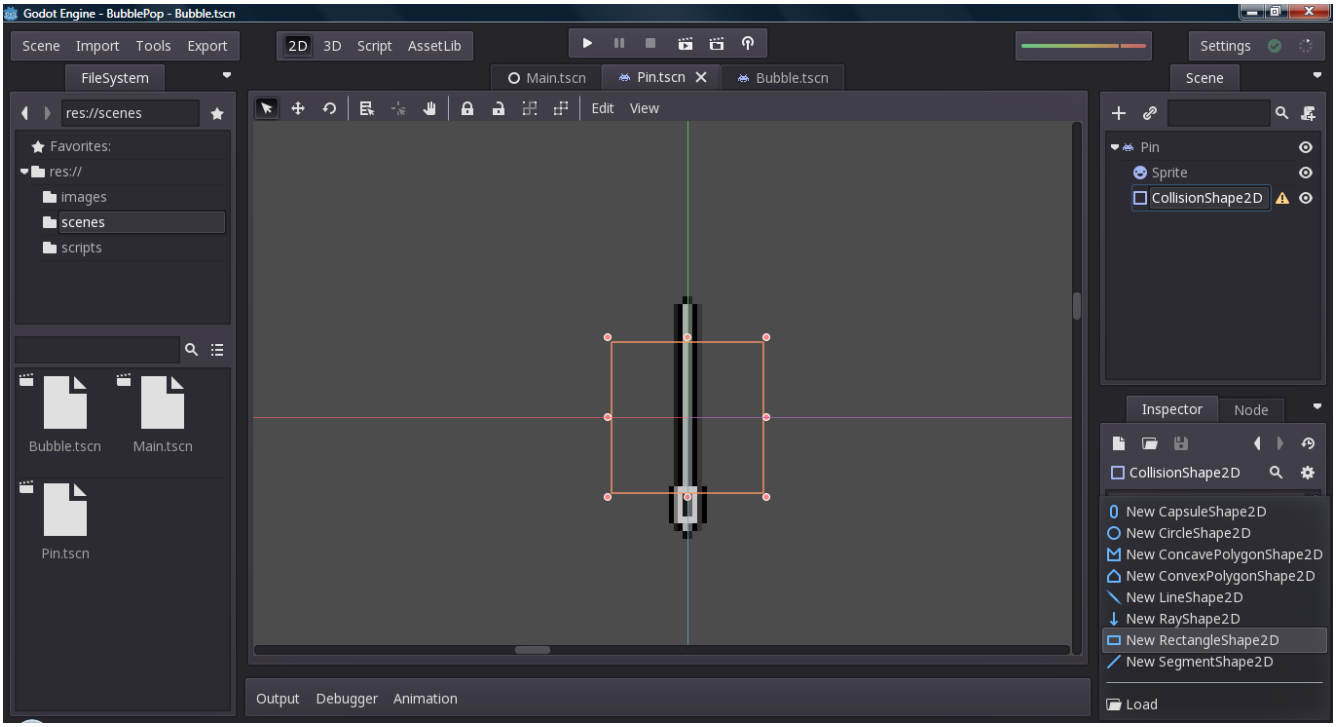
Now click the box beside the “Flags” property:



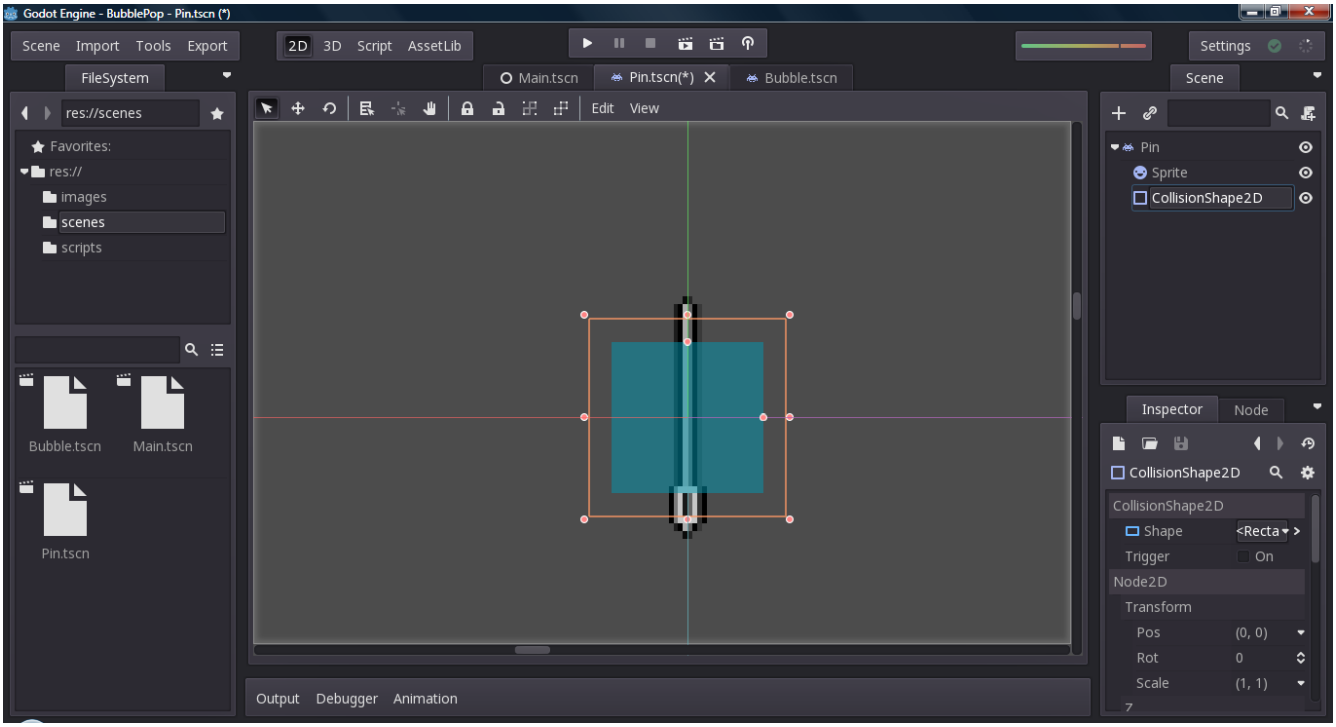
Turn off “Filter”:



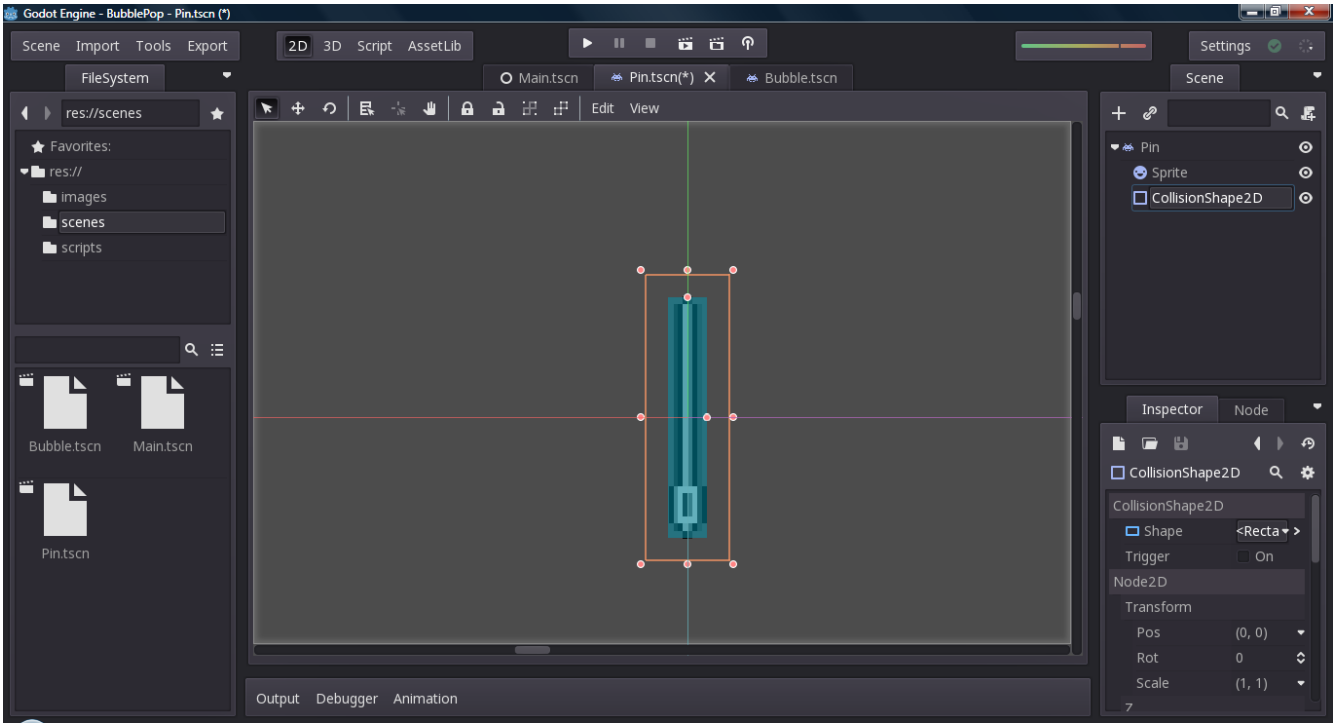
That’s better. We can do the same with our bubbles to improve their appearance as well. Don’t forget to also turn off filtering for the popping bubble texture. Now that we have fixed that slight issue, let’s set the shape for our pin:



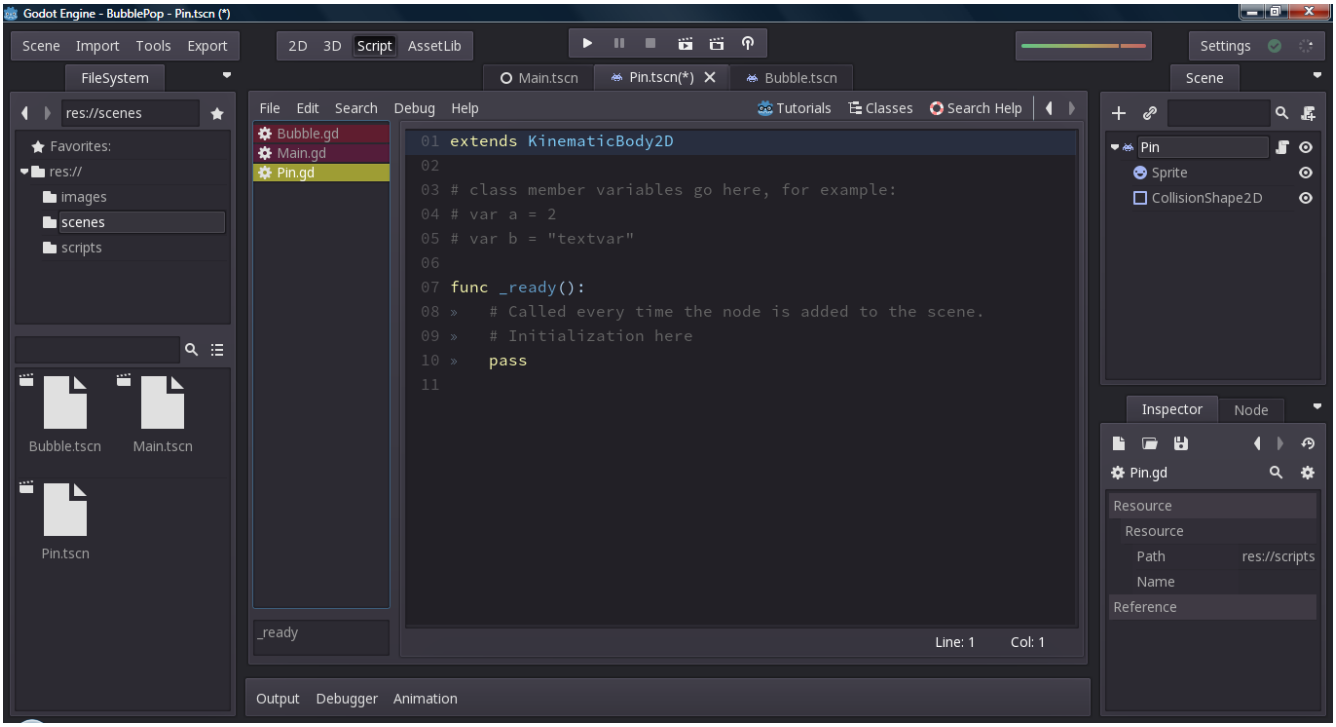
Since our pin is not round, we are going to choose “New RectangleShape2D” this time:



Now we need to adjust the rectangle with the little handles:



Next, we will attach a new script to our pin. Make sure you save the new script in our “scripts” folder:



Now it is time to start writing the logic for our pin. What we want it to do is follow the cursor and pop any bubble it touches. But how can we make our pin move with the cursor? We need to get the position of the mouse somehow. To do this, we will enable input event processing:

```
func _ready():
        #Enable event processing
        set_process_input(true)
```

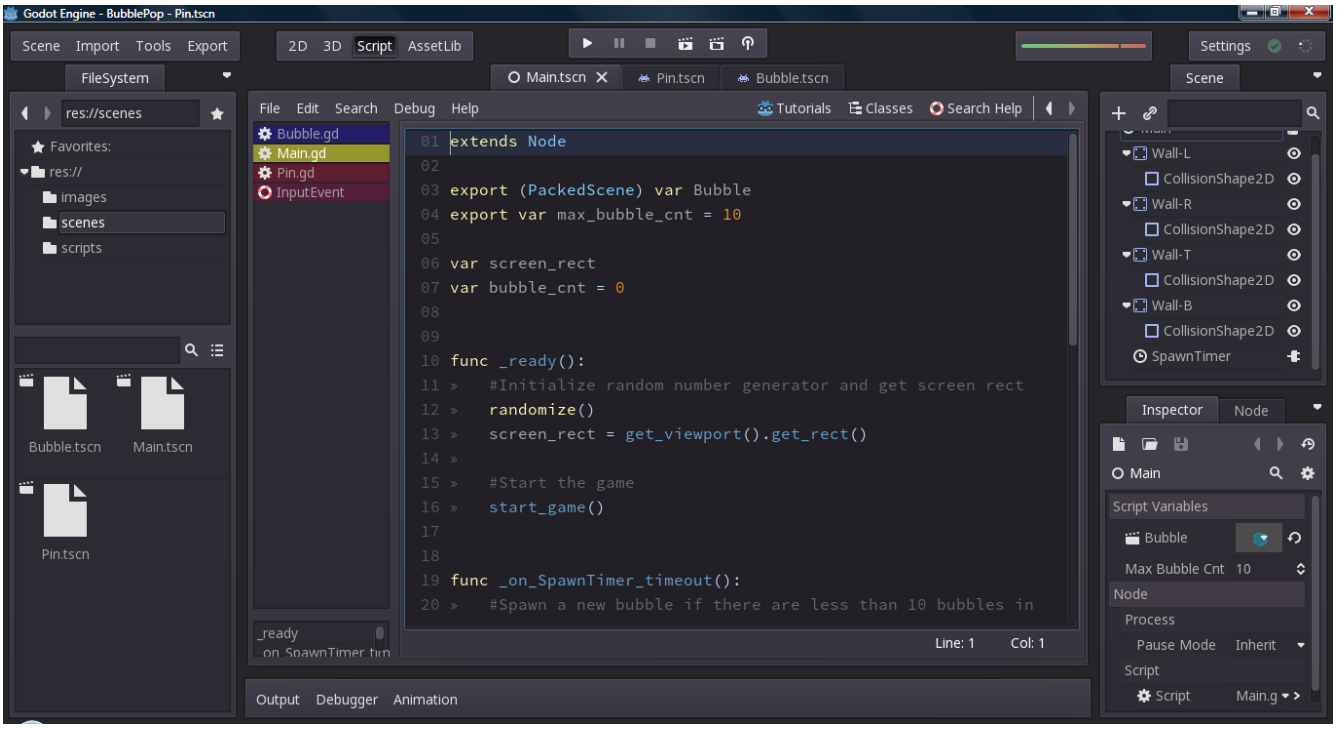
The "set_process_input" function is used to enable/disable input event processing. While input event processing is enabled on a node, Godot will call our "_input" function to handle the input event. Let's write our "_input" function now:

```
func _input(event):
        #Update pin position
        if event.type == InputEvent.SCREEN_DRAG:
            move_to(event.pos)
```

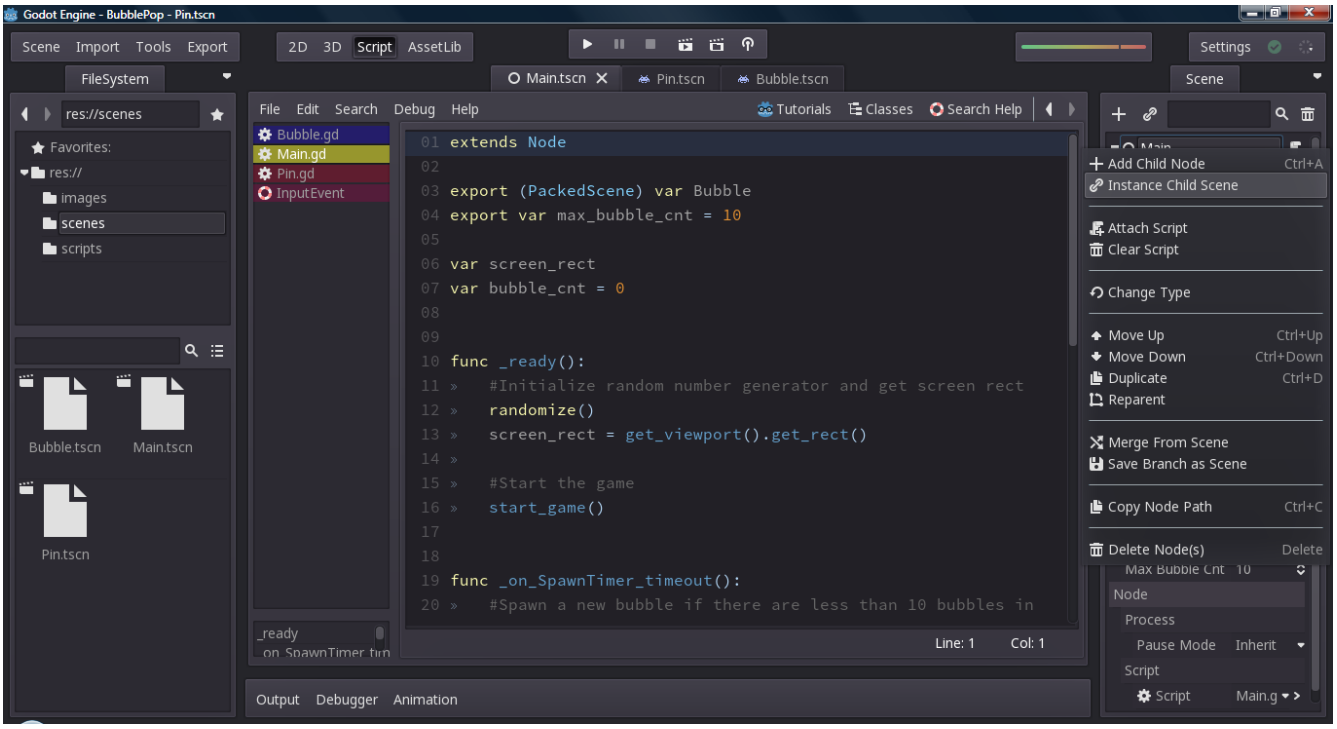
The "_input" function receives an InputEvent object as its first and only parameter. An input event can be one of many types such as pressing/releasing a key, moving the mouse, pressing/releasing a mouse button, moving a joystick, or pressing/releasing a joystick button. Therefore, we first must check what type of event we received. I bet you are wondering why I chose SCREEN_DRAG though. Whenever "Emulate Touchscreen" is turned on in our project settings, mouse events will be translated to touch events. This makes it much easier to support both PCs and mobile devices without writing more code. If the event is a screen

drag, we will simply move our pin to the position where the mouse (or your finger if you are using a mobile device) is on the screen.

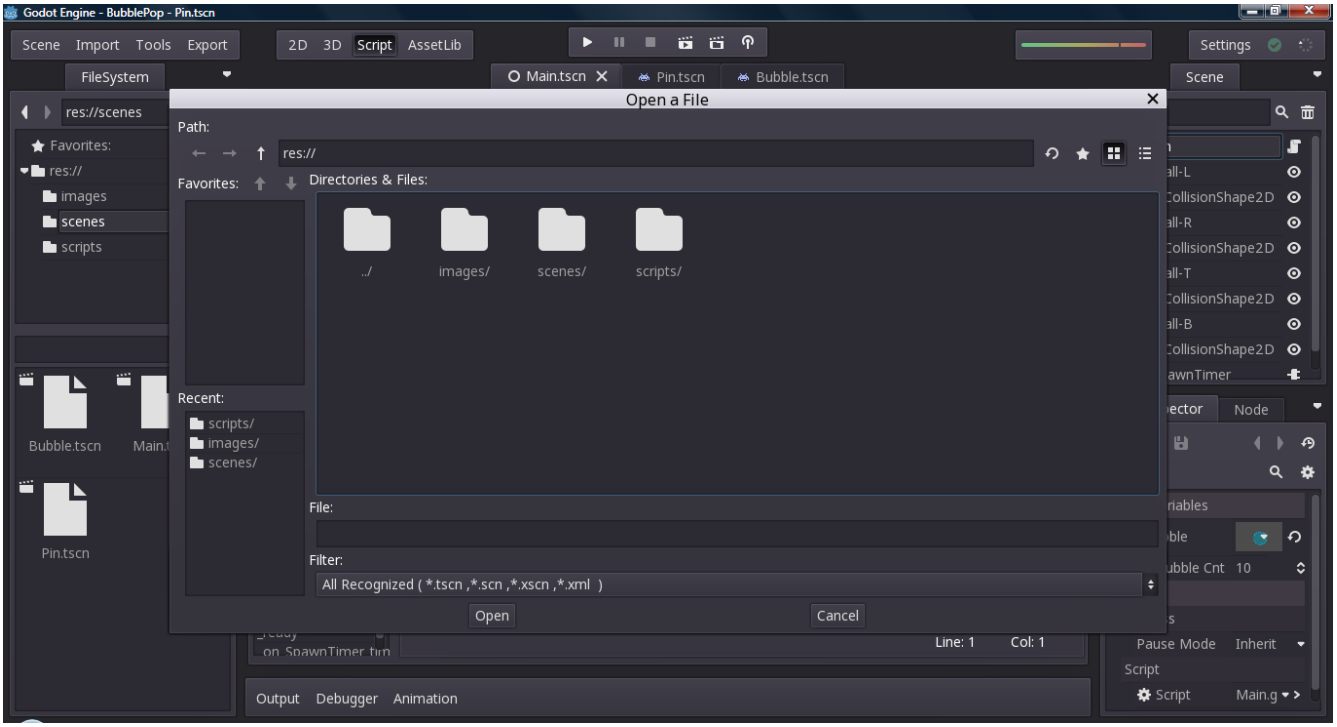
Now we need to switch back to our “Main” scene:



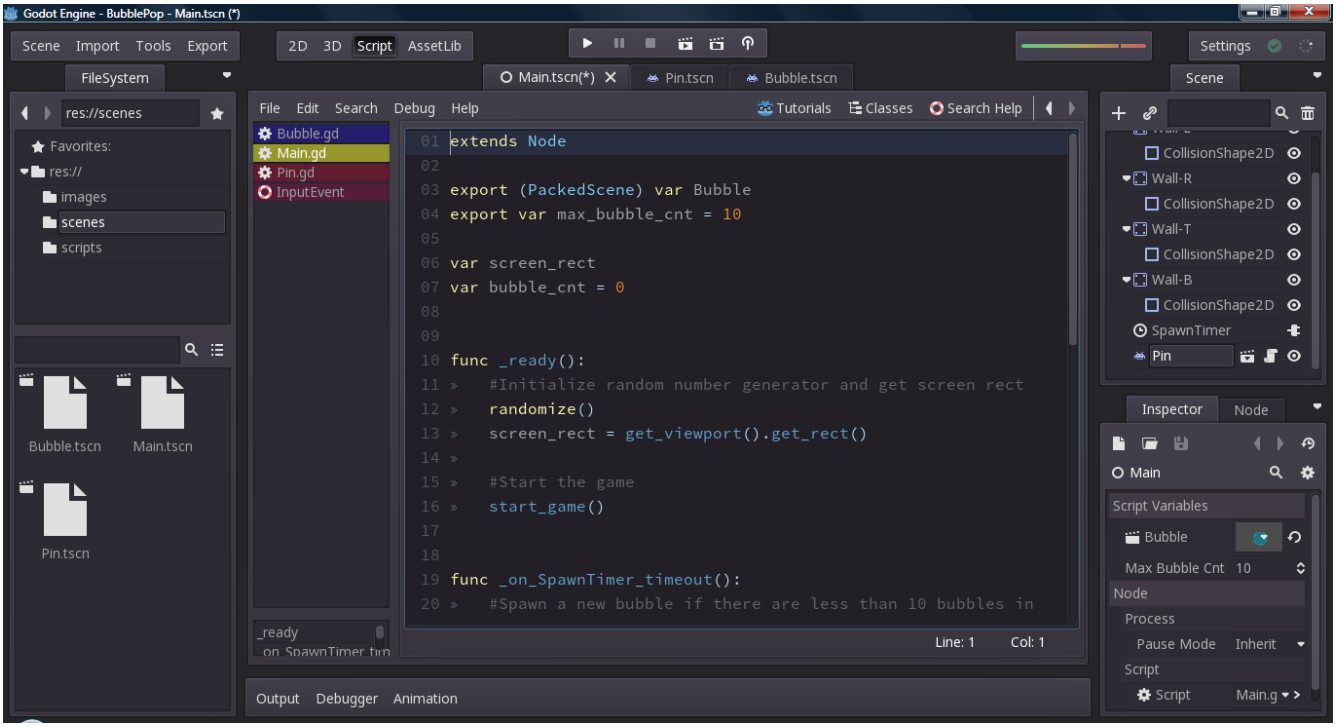
Since we will only ever need one instance of our pin, we will simply add it to the scene like any other node. Right-click the “Main” node:



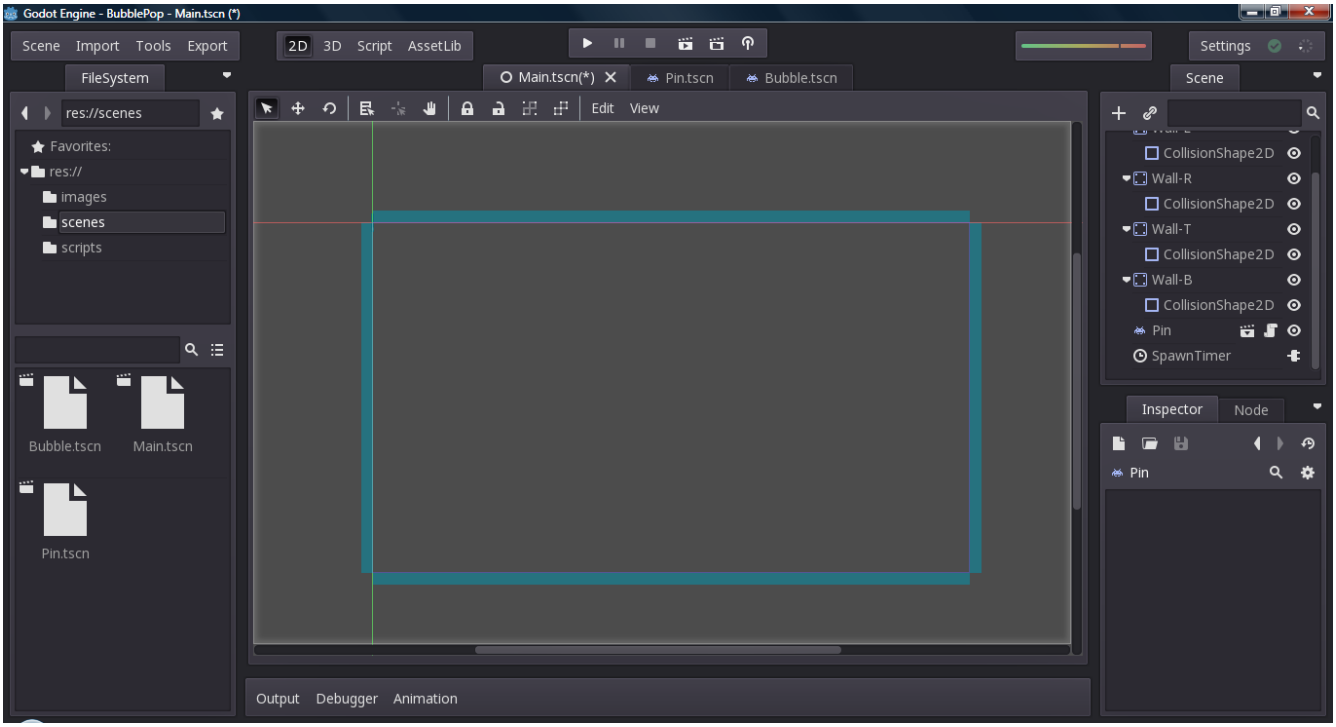
But this time, choose “Instance Child Scene”:



Now select your “Pin” scene:



And move the pin above the spawn timer:



Now let's try running our game again. Hmm… the pin will not move even if we move the mouse. Dragging will not work either. What is going on here? The reason why our pin will not move is because physics functions only work within the "_fixed_process" method. How can we move our pin then? Directly setting the pin's position is an option, however that would bypass all the physics for our pin. There is a better solution though. Switch back to your "Pin" scene and add a new variable under the first line like this:

```
var target_pos = Vector2()
```

This variable will be used to keep track of where the pin should go next on the screen. Now modify your "_input" function like this:

```
func _input(event):
	#Update pin target position
	if event.type == InputEvent.SCREEN_DRAG:
		target_pos = event.pos
```

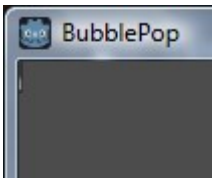
This will store the event position into our "target_pos" variable so we can use it from our physics code. Now we need to enable fixed event processing:

```
func _ready():
	#Enable event processing
	set_process_input(true)
	set_fixed_process(true)
```

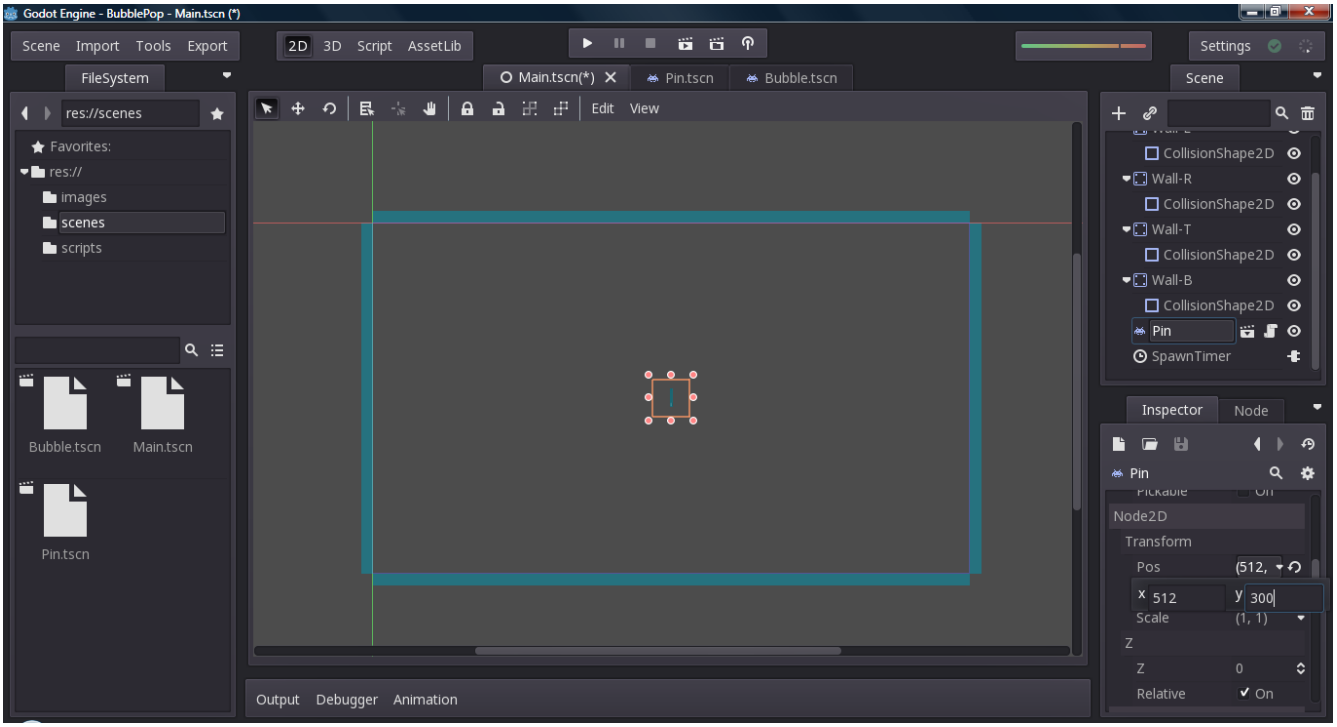
And last but not least, we will use our "_fixed_process" function to move the pin:

```
func _fixed_process(delta):
	#Update pin position
	move_to(target_pos)
```

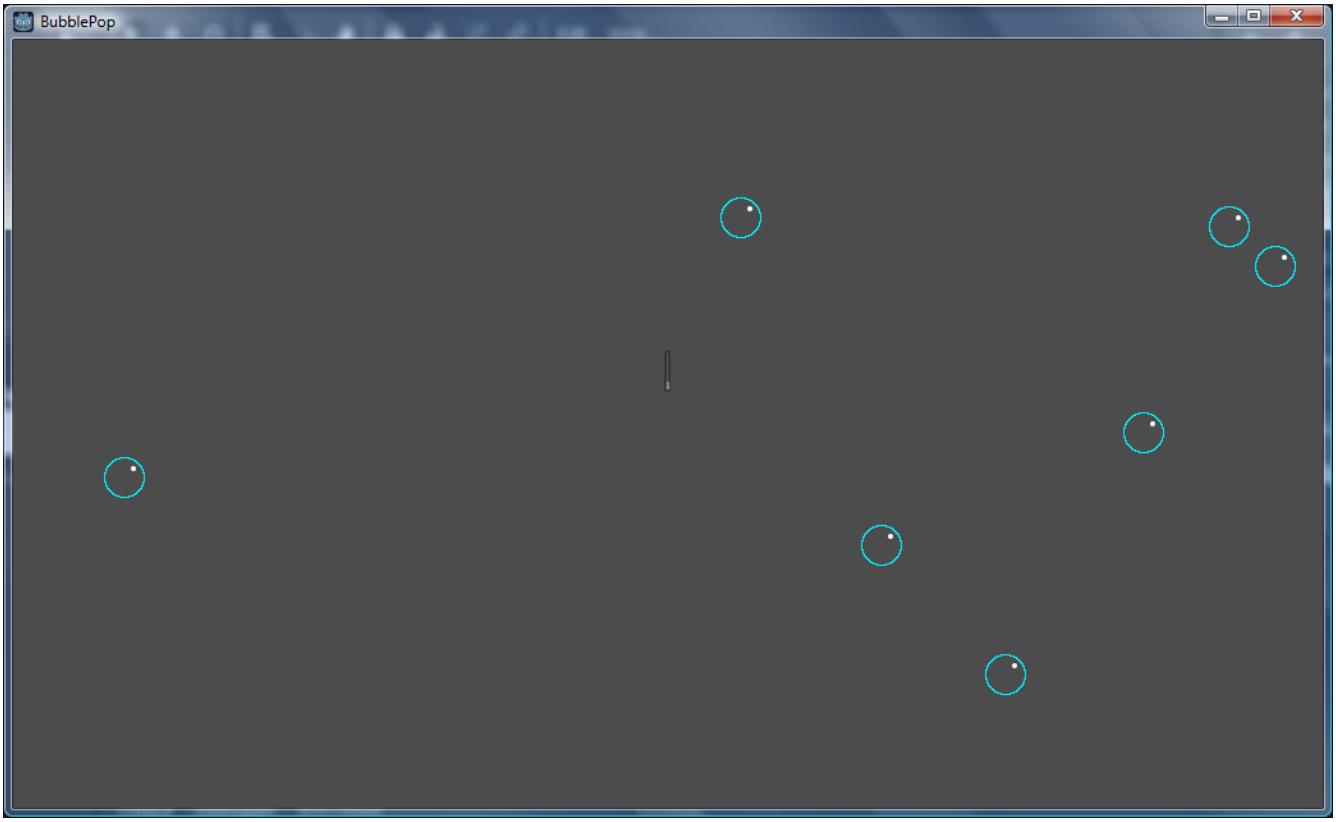
Now we can try running our game again. What!? The pin still refuses to move! How can this be? There is one last thing we have overlooked. If you look closely, you will notice that our pin is crammed in the upper left corner of the window and hanging partially outside it:



This is happening because we did not change the initial position of the pin. And since it is hanging outside the window, it is constantly colliding with the wall. Therefore, our pin cannot move at all. We can fix it by switching to the “Main” scene and moving our pin to the center of the window:



Now if we run our game again. The pin should move when we drag:



But there are still 2 slight bugs. For starters, our pin doesn't jump to the mouse position until we move the mouse. And also, our pin does not pop the bubbles. Let's go back to our "Pin" scene and fix this problem:

```
func _input(event):
        #Update pin target position
        if (event.type == InputEvent.SCREEN_TOUCH or
            event.type == InputEvent.SCREEN_DRAG):
            target_pos = event.pos
```

By adding an extra condition to the if statement in our "_input" function we can cause the target position of our pin to be set when we first press the mouse button (or touch the screen). Notice that we added parenthesis too. I did this so I could place each condition on its own line since together they are pretty long and hang off the edge of the script editor. If we test our game now, the pin should jump to the cursor when we click and follow it until we release the mouse button.

Next, we will add new code to our "_fixed_process" function:

```
func _fixed_process(delta):
	#Update pin position
	move_to(target_pos)

	#Pop any bubbles the pin touched
	if is_colliding():
	    var collider = get_collider()
	    var collider_name = collider.get_name()

	    if collider_name == "Bubble" or "@Bubble@" in collider_name:
	        collider.add_hp(-100)
```

We will start by checking if the pin collided with another object. If it collided, we will get the other object and its name. If the name matches a bubble, we will add -100 HP to the bubble, which will instantly pop it. Now if we run our game, we will be able to pop bubbles. However, wouldn't it be nice to have a sound effect play when the bubble pops? In the next lesson we will add one.