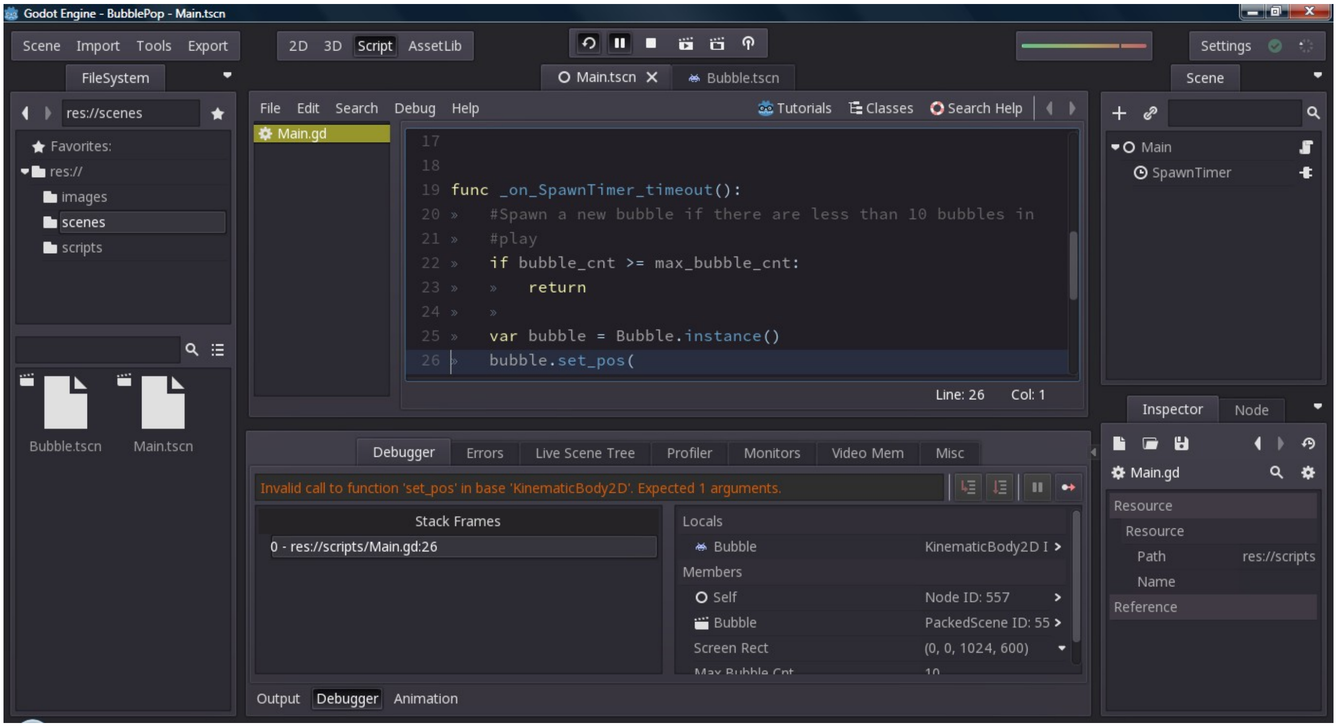
Godot 2D Game
Lesson 6: Debugging

Before we continue making our game, we will need to fix the bug we have at the moment. Here is a reminder of what it was:



```
func _on_SpawnTimer_timeout():
	#Spawn a new bubble if there are less than 10 bubbles in
	#play
	if bubble_cnt >= max_bubble_cnt:
		return
	
	var bubble = Bubble.instance()
	bubble.set_pos(
```

Line: 26 Col: 1

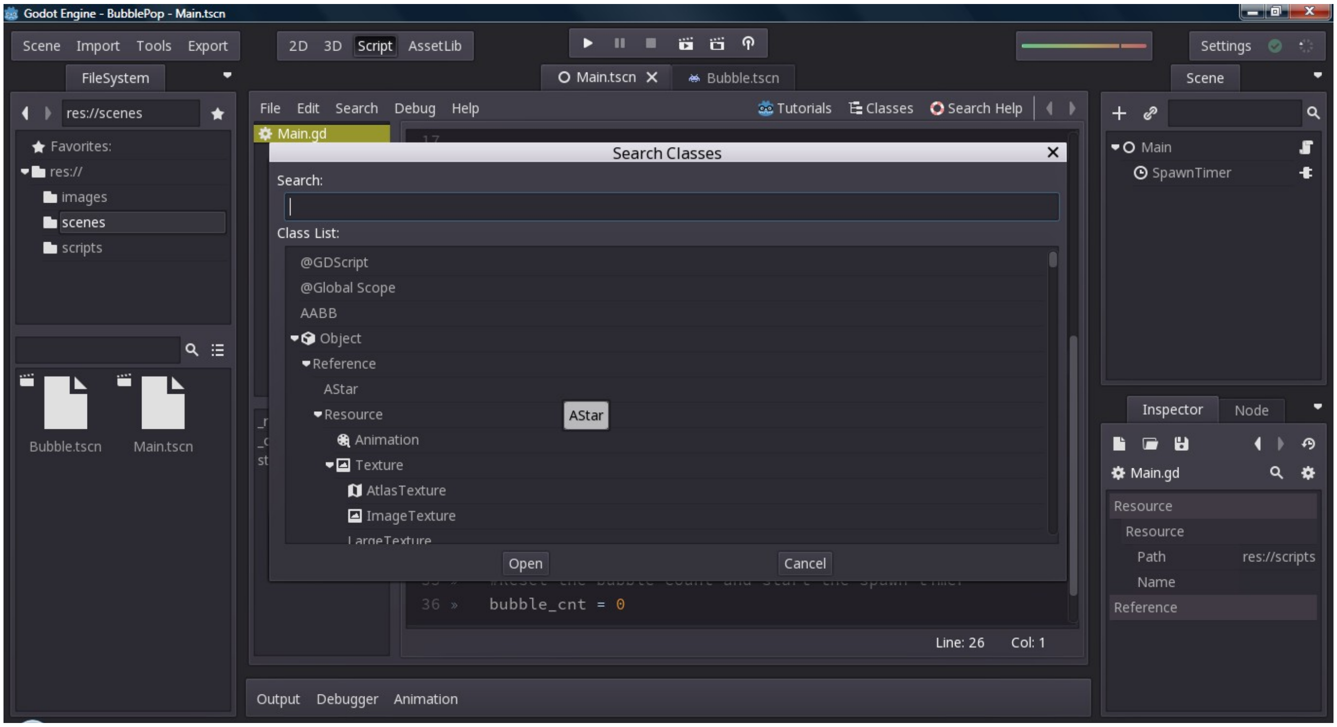
```
Invalid call to function 'set_pos' in base 'KinematicBody2D'. Expected 1 arguments.
```

Stack Frames
0 - res://scripts/Main.gd:26

Locals	
Bubble	KinematicBody2D I
Members	
Self	Node ID: 557
Bubble	PackedScene ID: 55
Screen Rect	(0, 0, 1024, 600)
Max Bubble Cnt	10

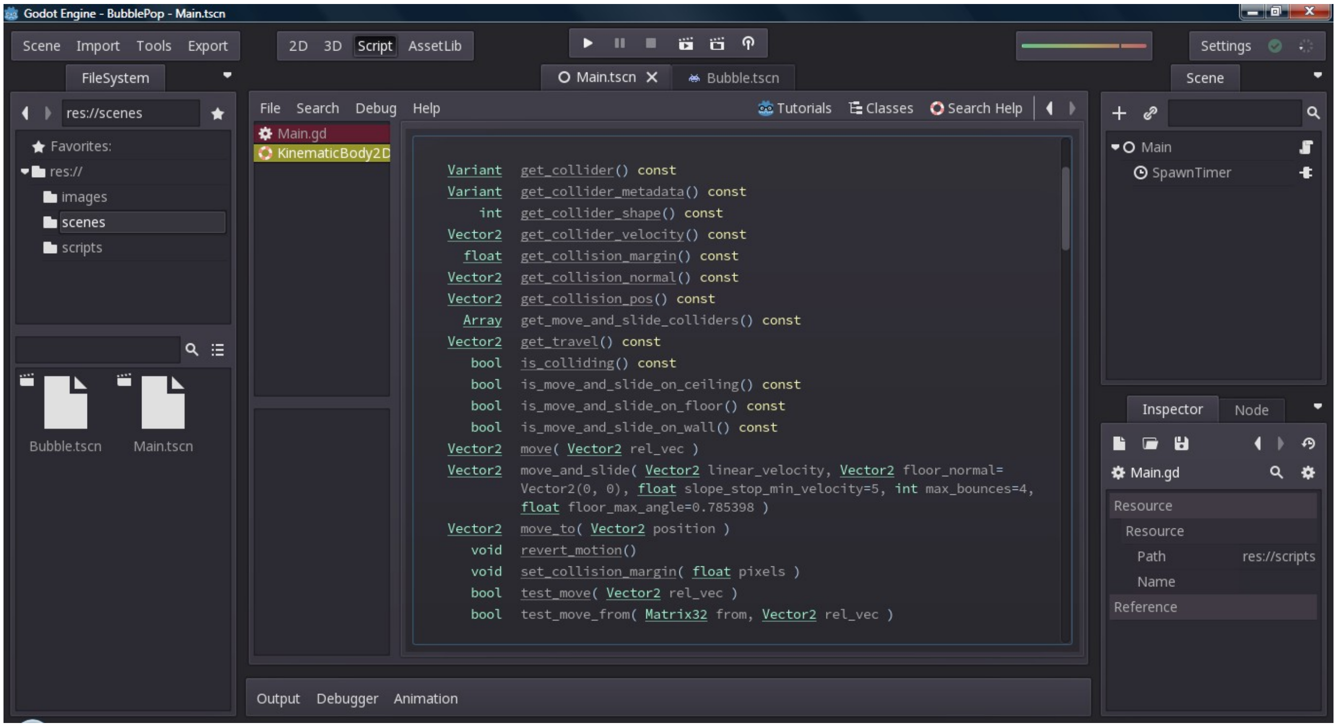
Main.gd
Resource
Resource
Path res://scripts
Name
Reference

Whenever our game crashes during testing, Godot will place the cursor on the line where the problem occurred and it will also display useful information about the state of the game when it crashed. Click the stop button above the middle workspace to close the game we were testing. And now, I will show you something else useful. Click on the “Classes” button:



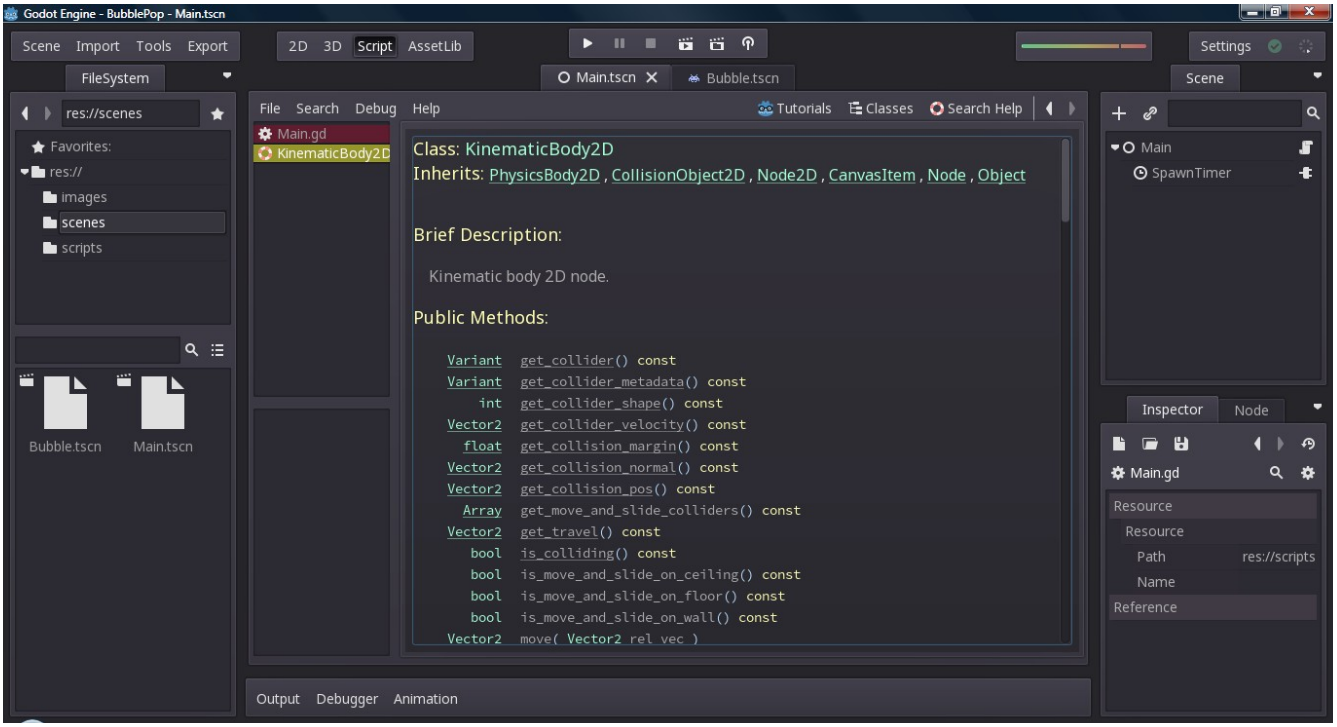
```
bubble_cnt = 0
```

Wow. What could this be? This dialog can be used to open documentation for any object in Godot. Let's take a look at the documentation for KinematicBody2D:



```
Variant  get_collider() const
Variant  get_collider_metadata() const
    int  get_collider_shape() const
Vector2  get_collider_velocity() const
  float  get_collision_margin() const
Vector2  get_collision_normal() const
Vector2  get_collision_pos() const
  Array  get_move_and_slide_colliders() const
Vector2  get_travel() const
   bool  is_colliding() const
   bool  is_move_and_slide_on_ceiling() const
   bool  is_move_and_slide_on_floor() const
   bool  is_move_and_slide_on_wall() const
Vector2  move( Vector2 rel_vec )
Vector2  move_and_slide( Vector2 linear_velocity, Vector2 floor_normal=
         Vector2(0, 0), float slope_stop_min_velocity=5, int max_bounces=4,
         float floor_max_angle=0.785398 )
Vector2  move_to( Vector2 position )
   void  revert_motion()
   void  set_collision_margin( float pixels )
   bool  test_move( Vector2 rel_vec )
   bool  test_move_from( Matrix32 from, Vector2 rel_vec )
```

If you take a look at the list of available functions, you will notice that "set_pos" is not listed here. But yet, if the function did not exist, we would've gotten a different sort of error. Scroll back to the top and read the "Inherits" line:



Class: KinematicBody2D
Inherits: PhysicsBody2D , CollisionObject2D , Node2D , CanvasItem , Node , Object

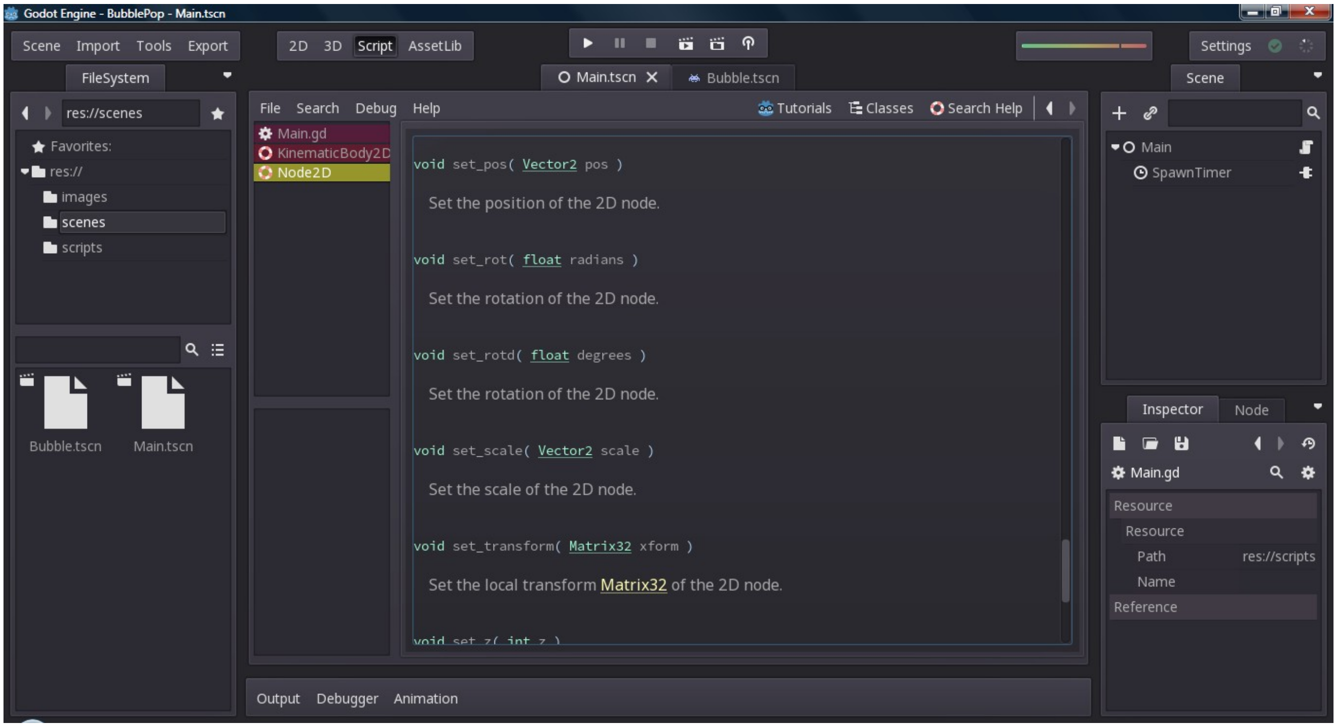
Brief Description:

Kinematic body 2D node.

Public Methods:

```
Variant  get_collider() const
Variant  get_collider_metadata() const
    int  get_collider_shape() const
Vector2  get_collider_velocity() const
  float  get_collision_margin() const
Vector2  get_collision_normal() const
Vector2  get_collision_pos() const
  Array  get_move_and_slide_colliders() const
Vector2  get_travel() const
   bool  is_colliding() const
   bool  is_move_and_slide_on_ceiling() const
   bool  is_move_and_slide_on_floor() const
   bool  is_move_and_slide_on_wall() const
Vector2  move( Vector2 rel_vec )
```

As you can see, this object inherits from PhysicsBody2D, CollisionObject2D, Node2D, CanvasItem, Node, and Object. Wow. That's a lot. But what does this mean? Whenever one object inherits from another object, the child object has all the functions and variables that the parent object has in addition to its own. If we look at the documentation for Node2D, we will see that it has a "set_pos" function:



```
void set_pos( Vector2 pos )
```

Set the position of the 2D node.

```
void set_rot( float radians )
```

Set the rotation of the 2D node.

```
void set_rotd( float degrees )
```

Set the rotation of the 2D node.

```
void set_scale( Vector2 scale )
```

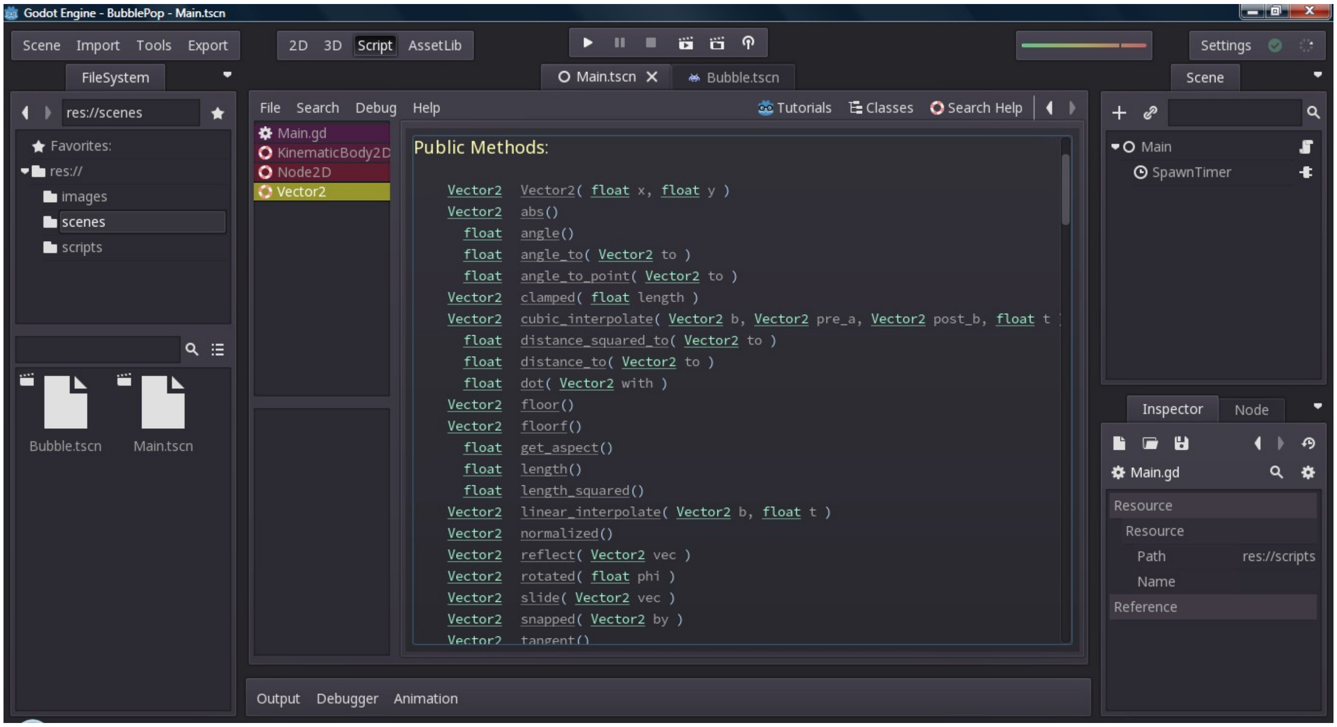
Set the scale of the 2D node.

```
void set_transform( Matrix32 xform )
```

Set the local transform Matrix32 of the 2D node.

```
void set_z( int z )
```

The “set_pos” function takes one parameter of the type Vector2. A Vector2 is a 2D position or direction. Now that we know this, we can take a look at the documentation for Vector2 to see how to use it:



Public Methods:

```
Vector2  Vector2( float x, float y )
Vector2  abs()
  float  angle()
  float  angle_to( Vector2 to )
  float  angle_to_point( Vector2 to )
Vector2  clamped( float length )
Vector2  cubic_interpolate( Vector2 b, Vector2 pre_a, Vector2 post_b, float t
  float  distance_squared_to( Vector2 to )
  float  distance_to( Vector2 to )
  float  dot( Vector2 with )
Vector2  floor()
Vector2  floorf()
  float  get_aspect()
  float  length()
  float  length_squared()
Vector2  linear_interpolate( Vector2 b, float t )
Vector2  normalized()
Vector2  reflect( Vector2 vec )
Vector2  rotated( float phi )
Vector2  slide( Vector2 vec )
Vector2  snapped( Vector2 by )
Vector2  tangent()
```

As you can see, we can create a new Vector2 by simply calling its Vector2 function. However, it works in a slightly different manner than most functions:

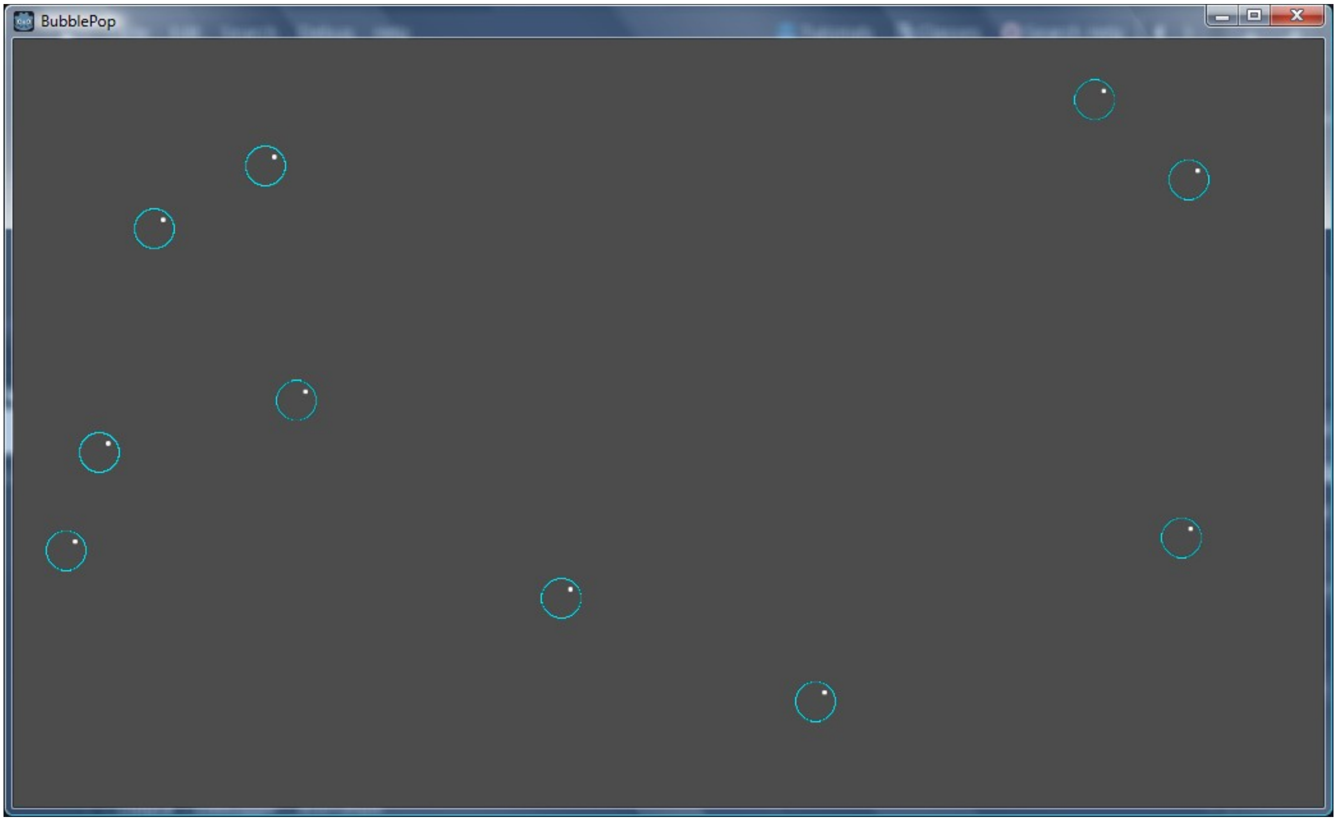
```
func _on_SpawnTimer_timeout():
	#Spawn a new bubble if there are less than 10 bubbles in
	#play
	if bubble_cnt >= max_bubble_cnt:
	return

	var bubble = Bubble.instance()
	var pos = Vector2(
	   rand_range(screen_rect.pos.x, screen_rect.end.x),
	   rand_range(screen_rect.pos.y, screen_rect.end.y)
	)
	bubble.set_pos(pos)
	add_child(bubble)
	bubble_cnt += 1
```

As you can see, the Vector2 function is called without using the usual “Object.method” syntax. This is because it is a special type of function called a constructor function. There are in fact 3 ways an object can be created:

```
Object()
Object.new()
Object.instance()
```

The method used to create a new object differs depending on the type of object. Most objects use the second method though. The first method is used mostly for simple objects such as vectors, the second object is used mostly for objects such as resources and nodes, and the third method is only used when instancing scenes.

To fix the error, we simply created a new Vector2 using our 2 random numbers and assign it to a new variable called “pos”. Then we pass “pos” as a parameter to “set_pos”. Now let’s try running our game again:



The bubbles spawn properly now and only 10 appear at a time. In the next lesson, we will modify our bubble so that it bounces around.