CODE NINJAS®

Bronze Belt Glossary & Documentation

Bronze Belt Glossary & Documentation

All new terms and concepts introduced in the Godot curriculum have been listed below. A definition/description is provided for each term/concept and images are provided where necessary.

Contents

Activity 00: Welcome to Godot

Child Node

A node that is the descendant of another node. Every node in a tree is a child node except the main root. These are like the branches of a tree.

GDScript

Godot's own scripting language similar to the language Python, saved as .gd files.

Input Map

A setting tab in Godot which allows the user to define inputs and set them to different keys or buttons.

Main root

The starting node of any project, like the root of a tree. Is the parent node to all other nodes in the project.

Main Scene

The main scene of the entire project saved from the Main root.

Node

The building blocks of a game that store data.

Parent node

A node that is a predecessor of another node and has at least one child node.

Project Path

The location where all projects are stored on the computer.

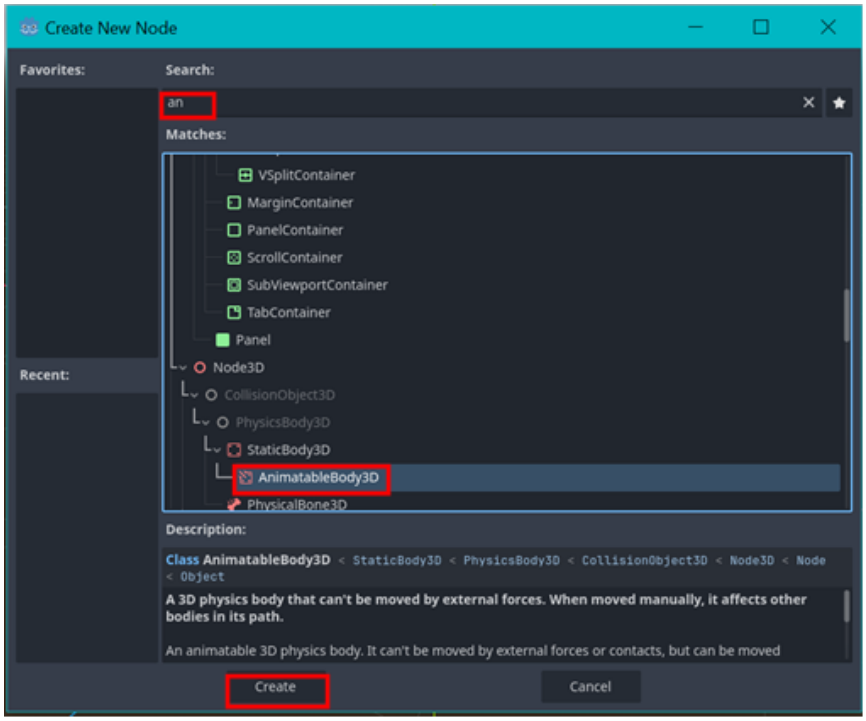
Tree Data Structure

A computer science method to store data in a hierarchical tree.

Activity 01: Dropping Bombs Part 1

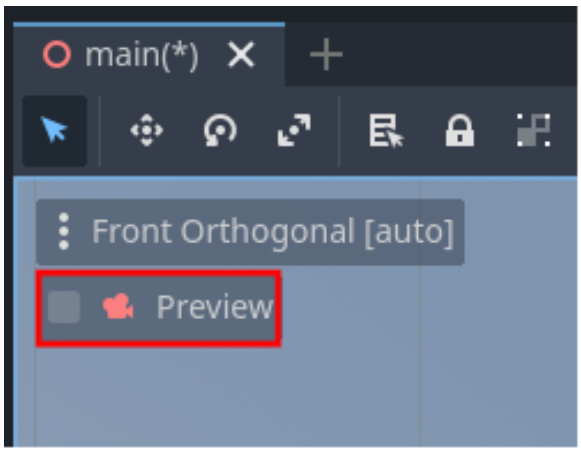
AnimatableBody3D

A 3D physics body node that cannot be moved by external forces but can be moved with inputs and scripts.



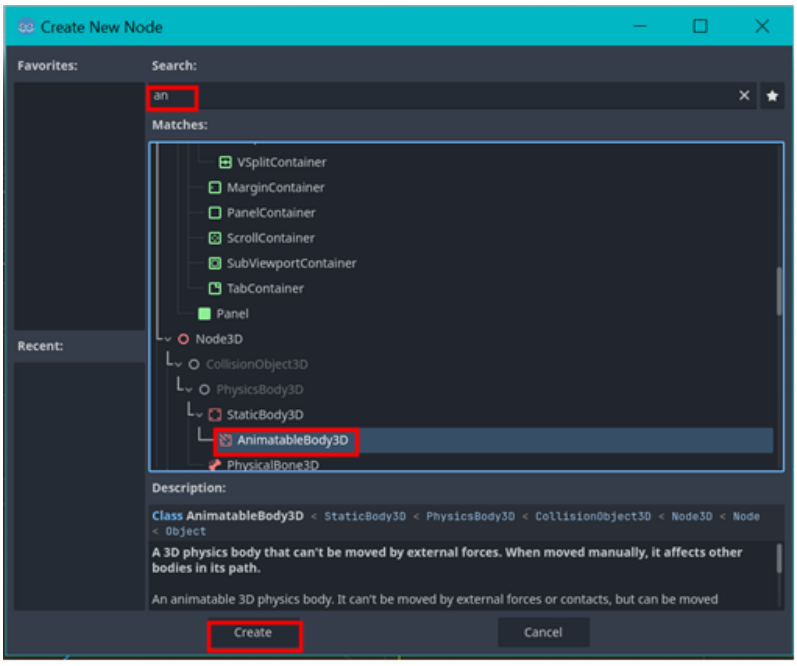
Camera3D

A 3D node that displays what is visible from its current location. Only one camera can be active per viewport.



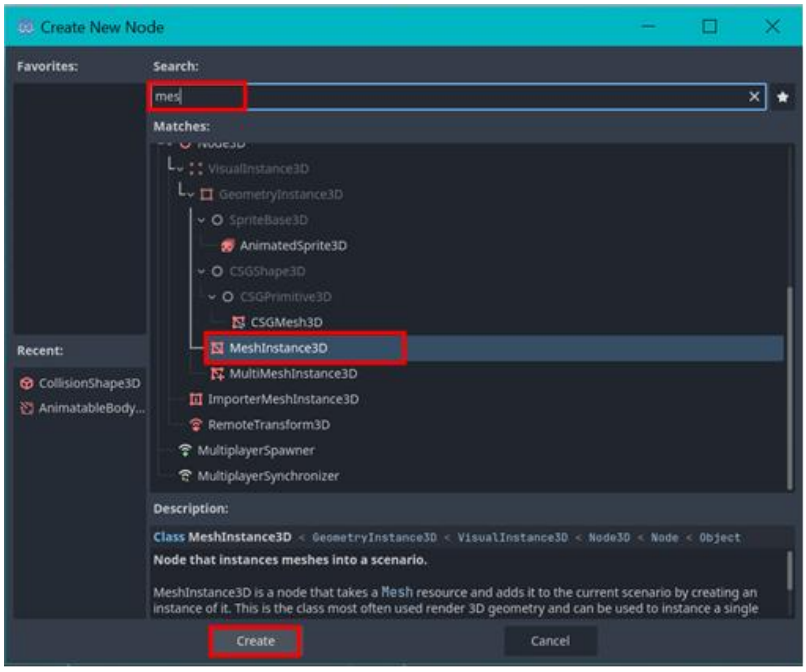
CollisionShape3D

A node that provides a collision shape to an Area3D or PhysicsBody3D node.



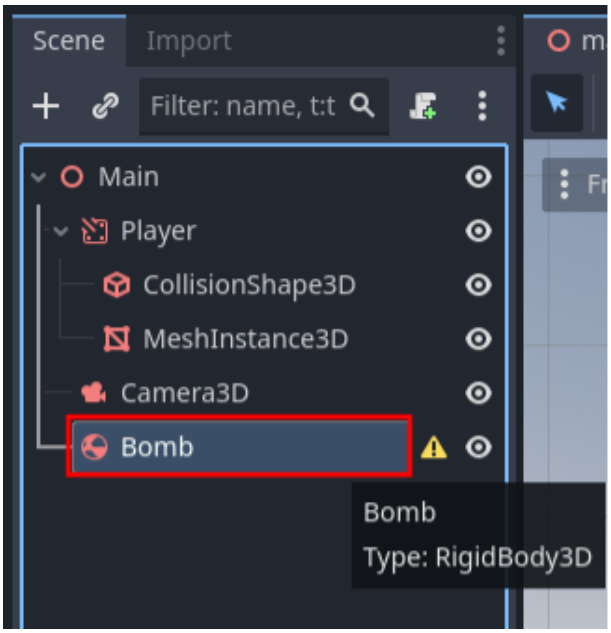
MeshInstance3D

A node that instances a mesh resource into a scene and can accurately generate CollisionShape3Ds based on a mesh.



RigidBody3D

A 3D physics body node that cannot be controlled directly but instead is controlled by forces.



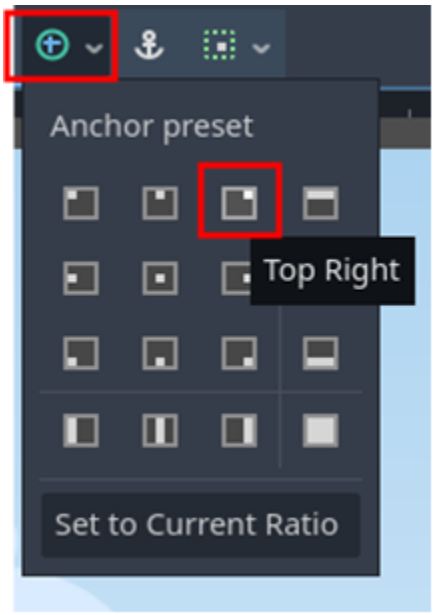
StandardMaterial3D

Default 3D material in Godot, designed to provide a wide range of features without requiring shader code.

Activity 02: Scavenger Hunt

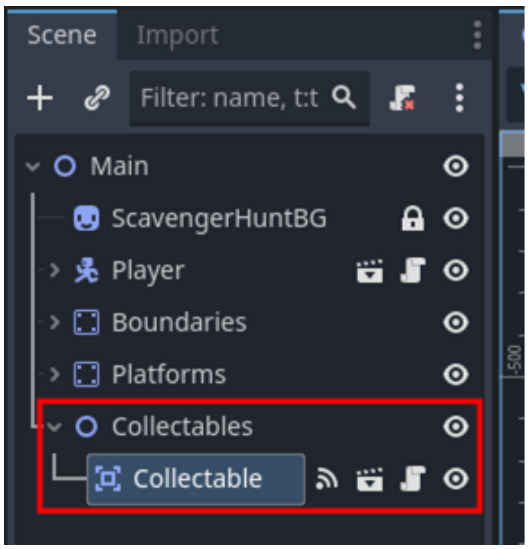
Anchor

At the top center of the editor, the anchor positions UI nodes at a set place on the screen (such as the top or left corner).



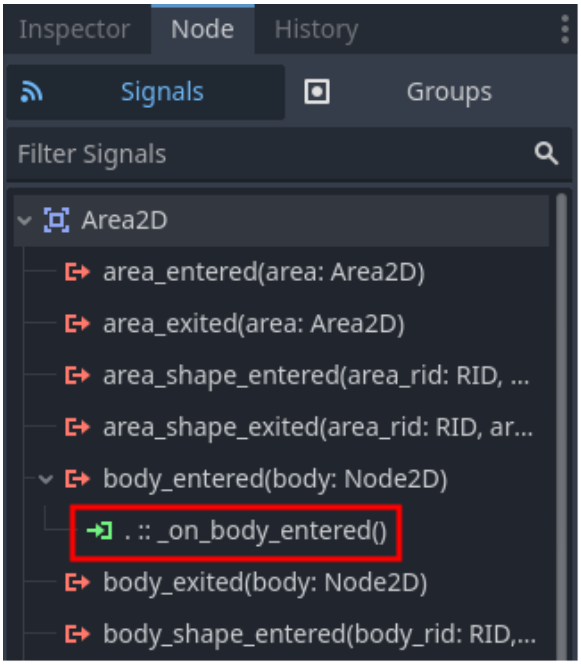
Area2D

A region of 2D space defined by one or multiple **CollisionShape2D** or **CollisionPolygon2D** child nodes.



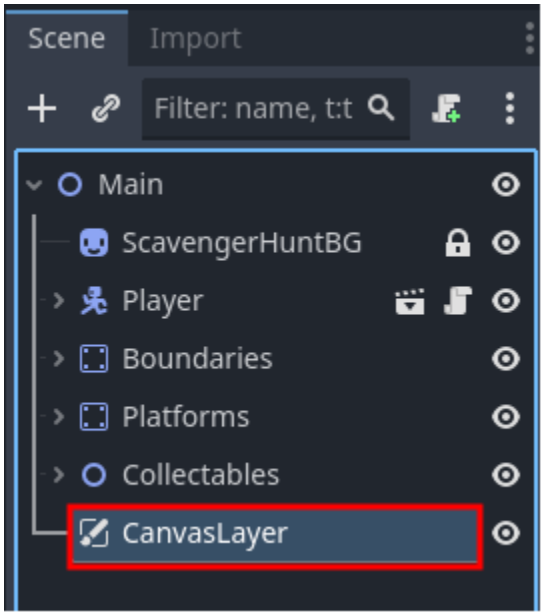
BODY_ENTERED SIGNAL

Like an **onOverlap** event in MakeCode, this signal will emit whenever a physics body enters collision with it.



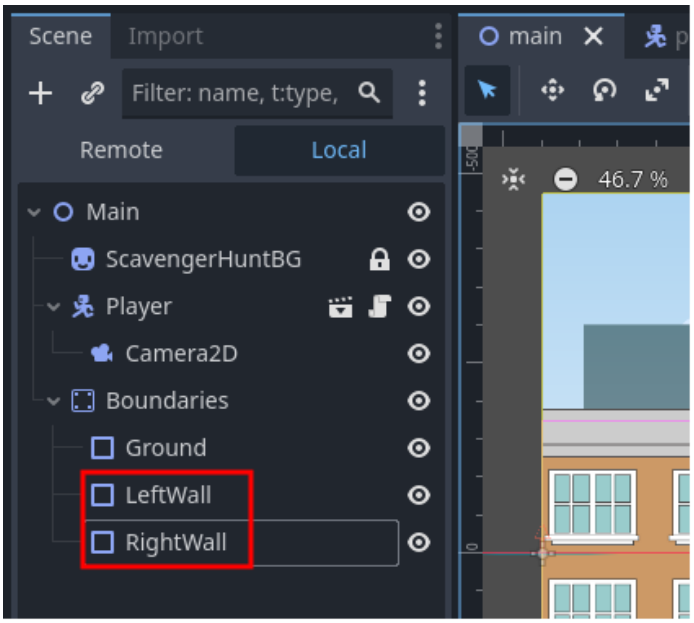
CANVASLAYER

This node makes sure that the UI stays in one place on the screen. Imagine a layer at the very top of the game window that's just for the UI; it receives information from the game, but the elements of the game cannot touch or move the UI.



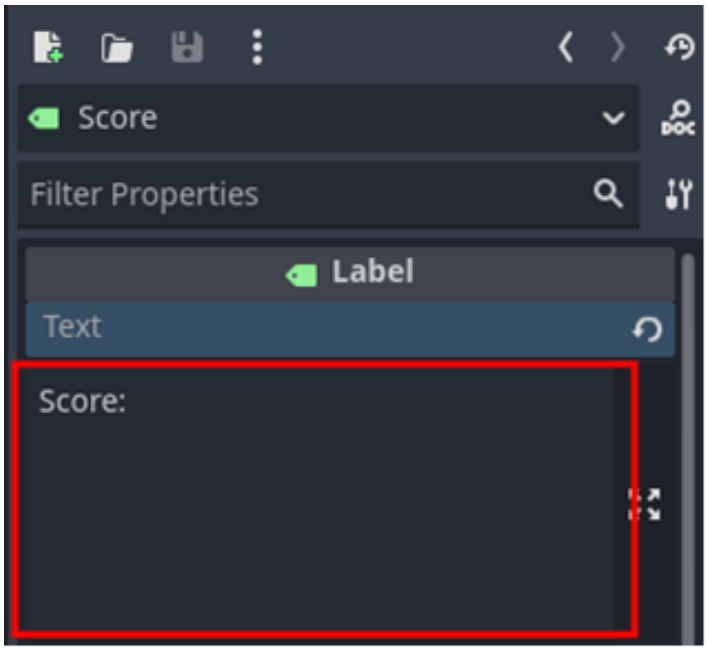
CollisionShape2D

A node that provides an editable Shape2D to a CollisionObject2D parent.



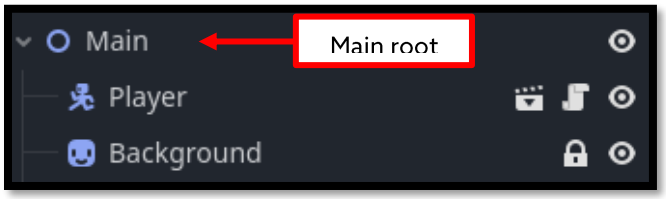
Label

A UI node used to display text on the screen.



Root

A root is always the node from which the scene was created. For example, the main scene being saved from the Main node makes it the Main root or the player scene being saved from the Player node makes it the Player root.

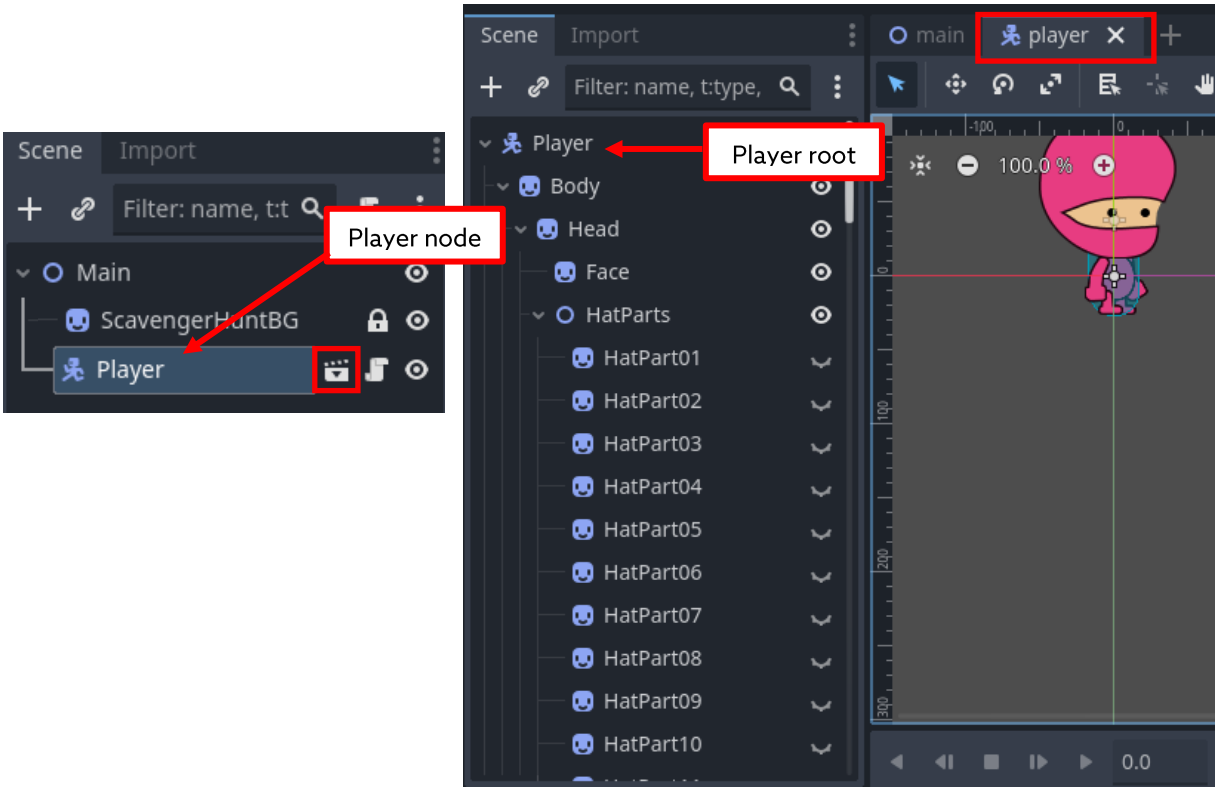


main scene

player scene

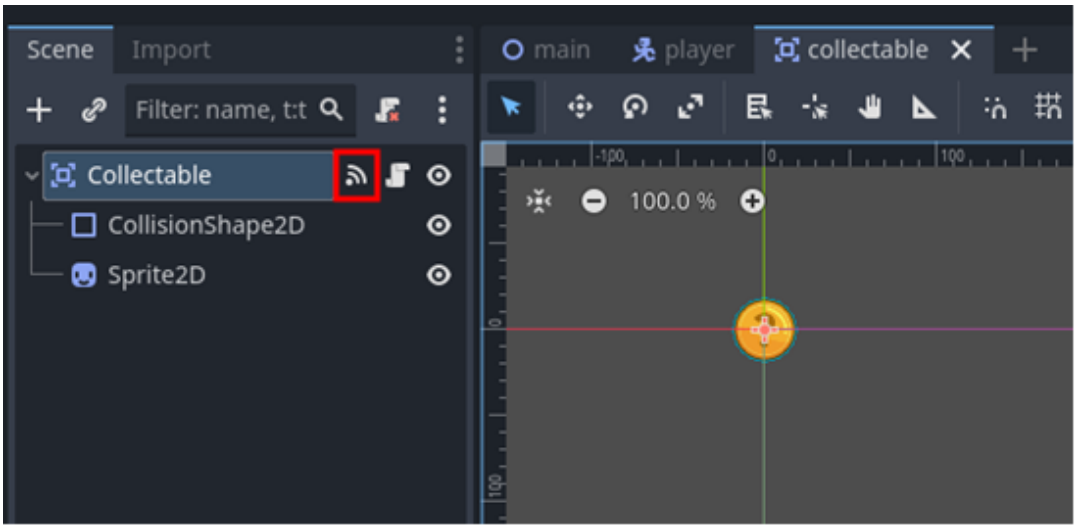
Scene

A tree of nodes saved to a .tscn file. Can be created from any node. When created from a child of the main root, it creates a sub-tree. Acts like a blueprint and can be reused many times throughout a game.



Signal

Messages that nodes send to each other when something specific happens to them. Signals are like events in MakeCode.

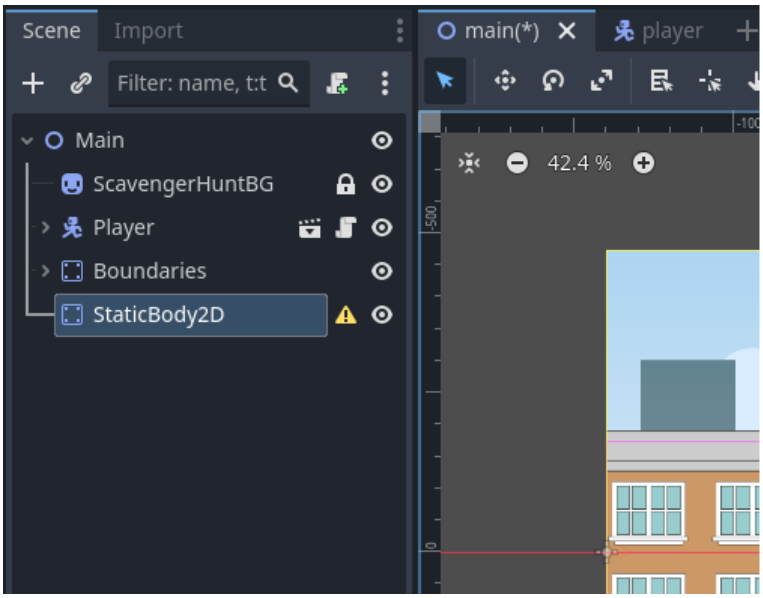


Sprite2D

A node that displays a 2D texture, such as an image.

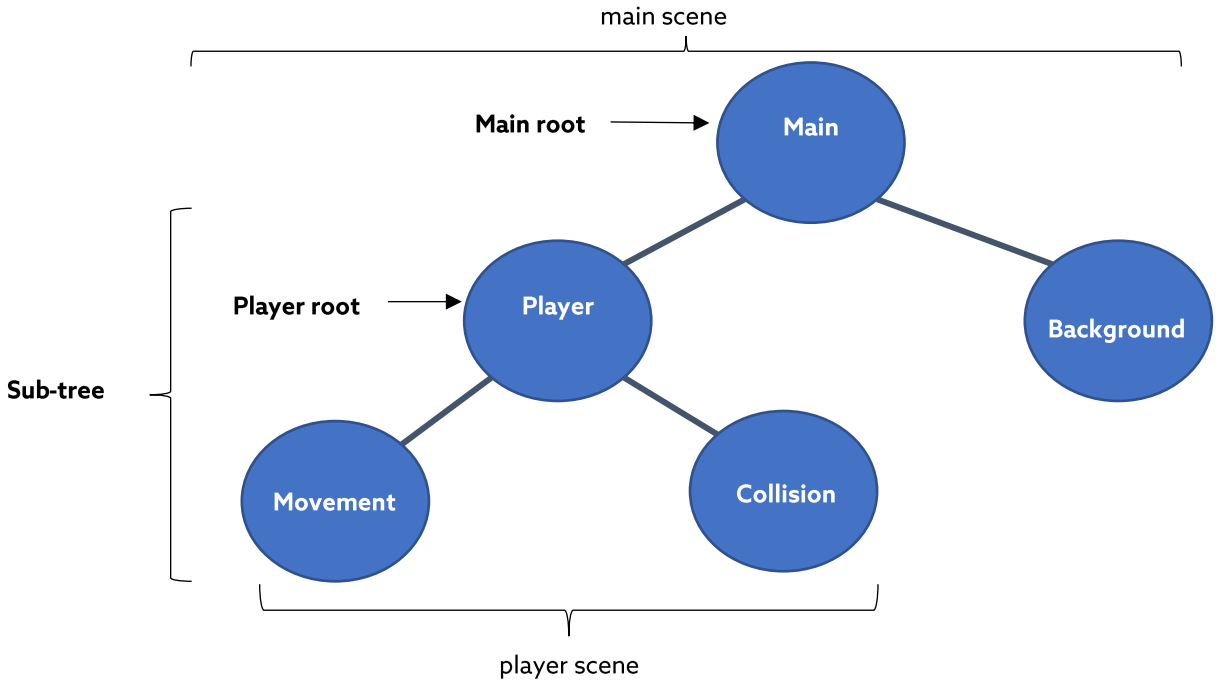
StaticBody2D

A 2D physics body that can't be moved by external forces.



Sub-Tree

A smaller section of the tree with its own root and children. Can contain a single node or multiple child nodes.

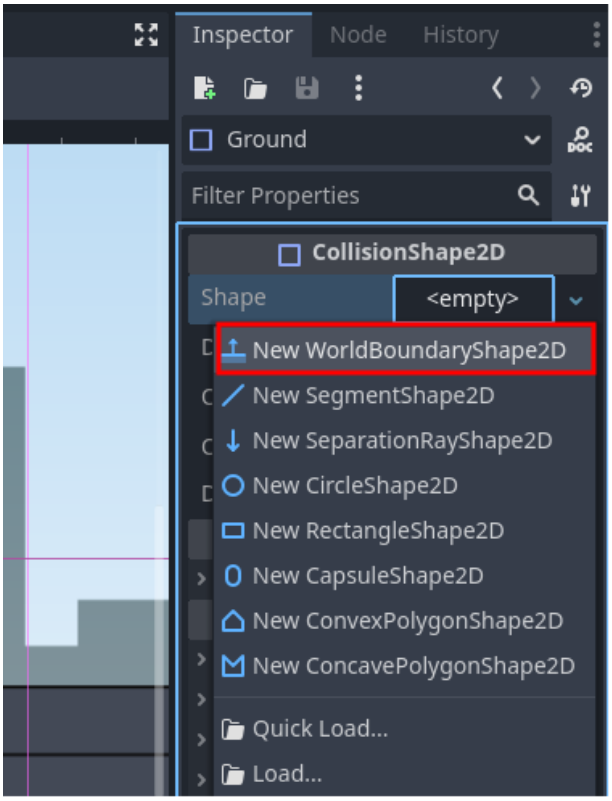


User Interface (UI)

Visual element that displays information to the user. This may include the score, health bar, or a game map.

WorldBoundary2D

A 2D collision shape that works like an infinite straight line that forces all physics bodies to stay above it.



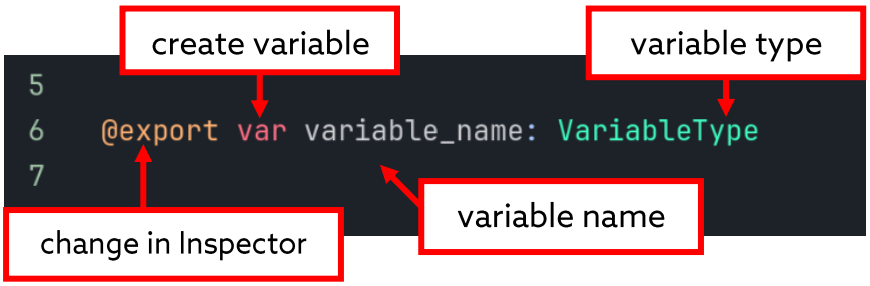
Activity 03: Meany Bird

Class

A **class** is like a blueprint for an object. All nodes in Godot are objects that have their own class, which allows each node to have its own set of properties.

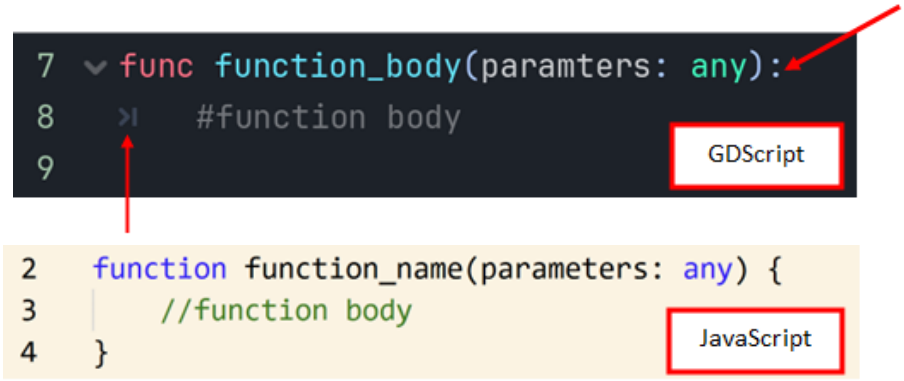
@export

Annotation used when declaring a variable so its value can be updated in the Inspector.



func

Keyword used to define a new function.



If-Statements

If-statements in GDScript are formatted similarly to functions in GDScript with a **colon** and **indentation** for the statement body, unlike JavaScript which uses curly brackets**.**

In GDScript, **parentheses ()** *can* be used but are not needed to hold the condition that needs to be true for the if-statement.

```
7    if (condition) {
8        // then
9    }
```
JavaScript

```
11   if condition:
12       #then
```
GDScript

Inheritance

Inheritance allows a node class to inherit properties and methods from other node classes.

Input

The keyword **Input** accesses the Input class which handles key presses, mouse buttons and movement, gamepads, and input actions. Actions and their events can be set in the **Input Map** tab in the Project Settings.

```
10  # --------------------
11  # TODO 2
12  # Create the _integrate_forces method
13  # --------------------
14  func _integrate_forces(state: PhysicsDirectBodyState2D) -> void:
15      if Input.is_action_just_pressed("click"):
16
17
```

INPUT.IS_ACTION_JUST_PRESSED()

`Input.is_action_just_pressed()`: returns true when the user has started pressing the action event in the current frame or physics tick. Will only return true in the frame or tick that the user pressed down the button. This is useful for code that needs to run only once when the action is pressed, instead of every frame while it's pressed.

Parameters:

1. `action` ***(StringName):*** the name of the action event.

Returns *(boolean):* whether the action was initiated in the current frame

_INTEGRATE_FORCES()

The `_integrate_forces()` method is called at the same time the engine calculates physics, which happens a set number of times per second.

Since rigid bodies are controlled by **Godot's built-in physics engine**, their properties can't just be updated since these changes may conflict with built-in physics. The `_integrate_forces()` method is used to "break" the physics engine and update properties (like applying velocity) to a rigid body.

`_integrate_forces()`: Part of the RigidBody2D/3D classes, this method is called during the physics processing step of the main loop. This method modifies the simulation state of the object.

Parameters:

1. `state` ***(PhysicsDirectBodyState2D/3D)*:** provides direct access to a physics body, allowing changes to physics properties.

Returns *(void)*: this method is void, it returns nothing.

LINEAR_VELOCITY

A property of RigidBody2D/3D; the speed at which an object changes its position in a straight line.

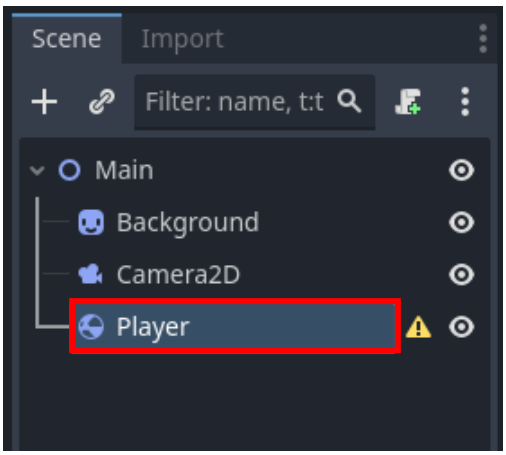
```
12   # Create the _integrate_forces method
13   # --------------------
14   func _integrate_forces(state: PhysicsDirectBodyState2D) -> void:
15       if Input.is_action_just_pressed("click"):
16           linear_velocity = Vector2.UP
17
```

METHOD

A **method** is a function that is defined in a class. These methods are like built-in functions that developers often use when working with a node class.

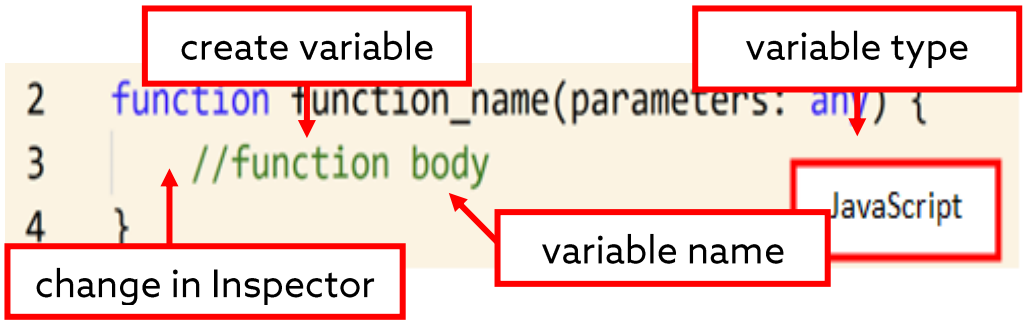
RIGIDBODY2D

A 2D physics body node that cannot be controlled directly but instead is controlled by forces.



VAR

Keyword used to declare a variable in GDScript.



VECTOR2.UP

`Vector2` creates a 2D vector with an x and y coordinate of 0 unless specified otherwise.

Using the constant `UP` creates a vector where **x = 0** and **y = -1**. Since the **positive Y** axis is **down** in 2D, a **negative Y** value will point **up**.

```
12  # Create the _integrate_forces method
13  # --------------------
14  func _integrate_forces(state: PhysicsDirectBodyState2D) -> void:
15      if Input.is_action_just_pressed("click"):
16          linear_velocity = Vector2.UP
17
```

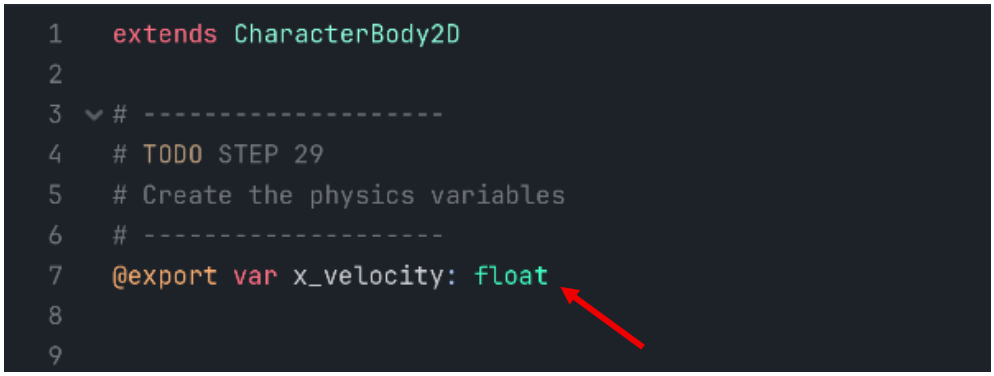
Activity 04: Sketch Head

CharacterBody2D/3D

A specialized physics body that cannot be moved by external forces but can be moved with inputs and scripts. Meant for complex movement, as opposed to AnimatableBody2D/3D.

FLOAT

`float` is a built-in type used for **floating-point** numbers, which are numbers that include decimal points, unlike an integer.



Gravity

Gravity is the force that pulls objects towards earth. The **force of gravity** is **constant** and does not change.

INPUT.IS_ACTION_PRESSED()

`Input.is_action_pressed()`: returns true or false depending on if the action event is being pressed in the current frame.

Parameters:

1. **`action`** ***(StringName):*** the name of the action event.

Returns ***(boolean):*** whether the action event is pressed

```
12  # --------------------
13  func _process(delta: float) -> void:
14      velocity.x = 0
15
16      if Input.is_action_pressed("ui_left"):
17          |
18
```

IS_ON_FLOOR()

`is_on_floor()`: Part of the CharacterBody2D/3D classes, this method returns true if the physics body collided with the floor on the last call of the **`move_and_slide()`** method.

Parameters: None

Returns ***(boolean):*** whether the CharacterBody2D/3D is on the floor

```
19  func _physics_process(delta: float) -> void:
20      velocity.y += gravity * delta
21
22      if is_on_floor():
23          
```

MOVE_AND_SLIDE()

`move_and_slide()`: Part of the CharacterBody2D/3D classes, this method moves the physics body based on velocity. If the physics body collides with another physics body (like a platform), it will redirect its velocity to slide along the surface of the object instead of stopping immediately.

Parameters: None

Returns *(boolean):* whether a collision happened

```
19  func _physics_process(delta: float) -> void:
20      velocity.y += gravity * delta
21
22      if is_on_floor():
23          velocity.y = down_speed
24
25      move_and_slide()
```

_PHYSICS_PROCESS()

`_physics_process()`: This method is called during the physics processing step of the main loop. Physics processing allows for the frame rate to be synced to the game physics.

Parameters:

1. **`delta` *(float):*** the time in seconds since the previous frame. Delta will *generally* be constant in this method.

Returns *(void):* this method is void, it returns nothing.

```
19  func _physics_process(delta: float) -> void:
20      velocity.y += gravity * delta
21
```

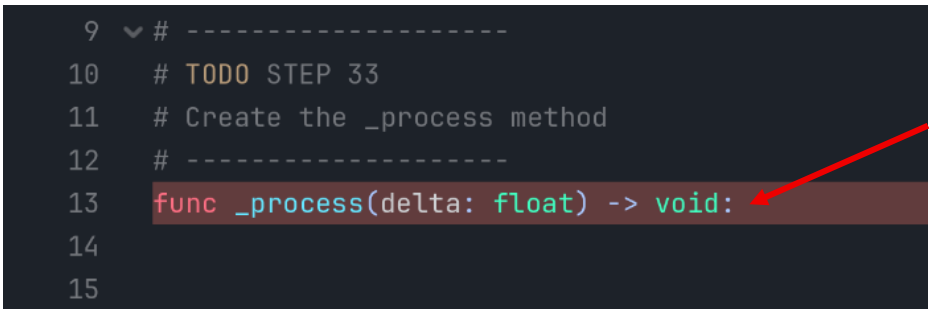
_PROCESS()

`_process()`: Part of the Node class, this method is called during the processing step of the main loop. Processing happens at every frame and as fast as possible.

Parameters:

1. **`delta`** ***(float):*** the time since the previous frame, in seconds. Since processing happens as fast as possible, delta is not constant.

Returns *(void):* this method is void, it returns nothing.

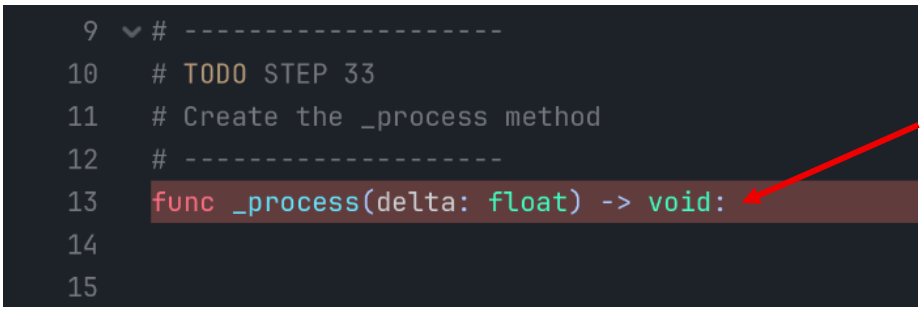


VELOCITY

A property of CharacterBody2D – The current speed of the velocity vector.

VOID

`-> void` shows what the method returns. This method is **`void`**, meaning it does not return anything.



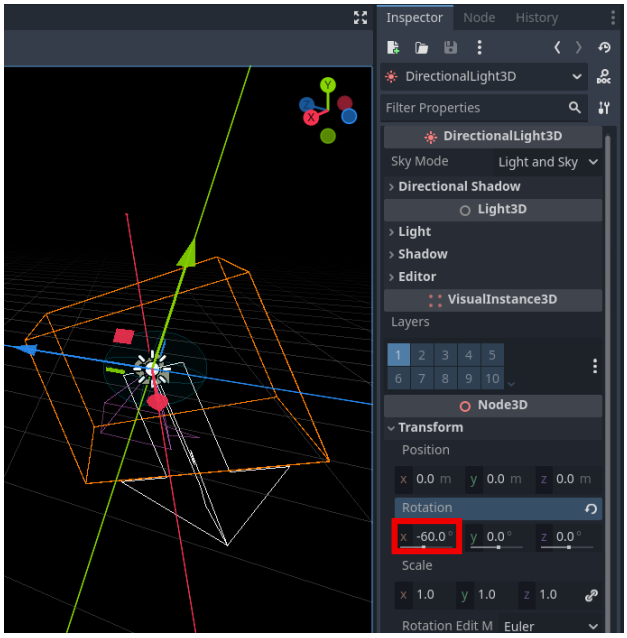
Y-VELOCITY

Speed in the y-axis (up and down). The y-velocity is what allows an object to jump/move up against gravity.

Activity 05: Don't Touch the Cubes

DirectionalLight3D

A 3D light emitting node that casts an infinite sheet of light rays across the entire scene based on the DirectionalLight3D's rotation values.



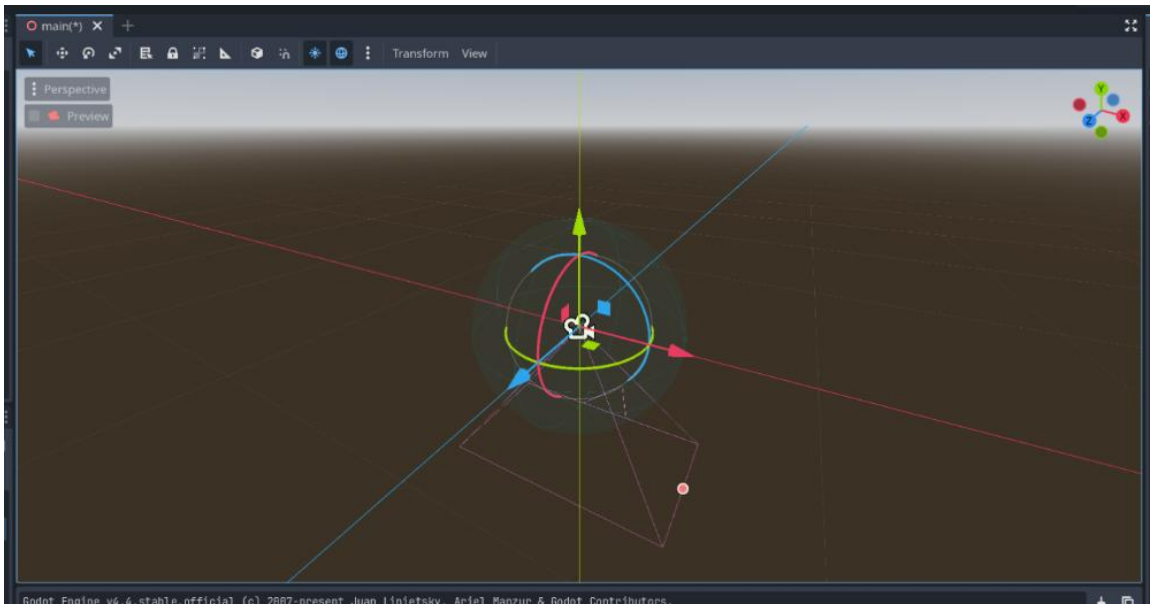
EXTENDS

A keyword typically followed by the path to another script. Acts as if the other script's code is pasted at the top of the current script but hides it.

```
1  extends "res://Scripts/player_base.gd"
2
3  func handle_movement(delta: float) -> void:
4      # --------------------
```

FRUSTUM

A Frustum is the shape that illustrates what a camera can see! It is shown by the pink lines that extend out of Camera3D nodes in Godot. The area enclosed by the pink lines may seem small, but the Camera3D node has “Near” and “Far” parameters that determine which objects can be seen by the camera. Notice how large the “Far” parameter is compared to “Near”! The frustum stretches much further than what the preview in Godot shows.



GLOBAL_BASIS

A property of Node – The rotation and scale information relative to world space.

GLOBAL_POSITION

A property of Node – The position information relative to world space.

NORMALIZED()

`normalized()`: Part of the Vector2/3 classes, this method returns the normalized scaling of the given vector so the length is 1.

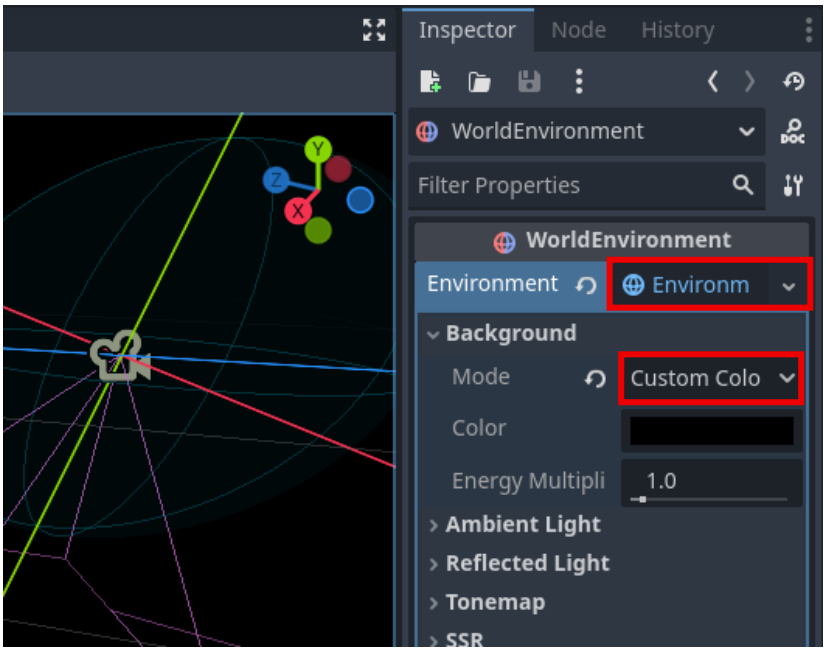
Parameters: None

Returns *(Vector)*: normalized scaling of the target of length 1.

```
18  # --------------------
19  # TODO STEP 71
20  # Normalize input direction so that it's a unit vector
21  # --------------------
22  input_dir = input_dir.normalized()
```

WORLDENVIRONMENT

A node that determines how the world is rendered and grants access to advanced visual effects like glow, fog, and reflections.



ACTIVITY 06: SUPERSHAPES

ADD_CHILD()

A method used to add a node as a child to another node in the scene tree, making it appear in the game.

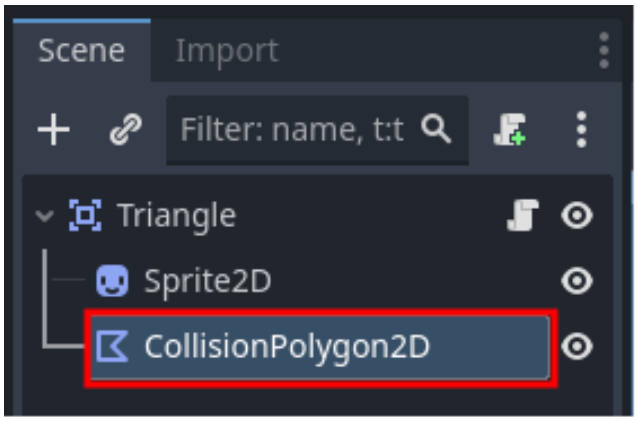
AWAIT

A keyword in GDScript that pauses code execution until a specific signal is received. It's often used with timers or other events to wait for something to finish.

```
16  func spawn():
17      # ---------------------
18      # TODO 3
19      # Wait to spawn the first shape
20      # ---------------------
21      await get_tree().create_timer(spawn_delay).timeout
```

COLLISIONPOLYGON2D

A node that provides a polygon shape to a CollisionObject2D parent and create custom polygon collision shapes.



CREATE_TIMER()

`create_timer()`: part of the SceneTree class, creates and starts a temporary **Timer** node that runs for the specified number of seconds. It is added to the game and is removed after it finishes.

Parameters:

1. `time_sec` ***(float)*****:** duration of the timer in seconds
2. `pause_mode_process` ***(bool, optional)*****:** if **true,** timer will pause when the game is paused

Returns *(node)*: A temporary **Timer** node

```
16  func spawn():
17      # --------------------
18      # TODO 3
19      # Wait to spawn the first shape
20      # --------------------
21      await get_tree().create_timer().timeout
22          SceneTreeTimer create_timer(time_sec: float, process_always: bool = true
```

GET_TREE()

A method that returns the current **SceneTree**, which manages scenes, nodes, and groups. It's needed to access game-wide features like timers or changing scenes.

```
16  func spawn():
17      # --------------------
18      # TODO 3
19      # Wait to spawn the first shape
20      # --------------------
21      await get_tree().create_timer(spawn_delay).timeout
```

INPUT.GET_AXIS()

A method that checks if two inputs are being pressed (held) and returns a number to show the direction of that input.

`Input.get_axis()`: checks if two inputs are being pressed and returns a number to show the direction of that input.

Parameters:

2. `negative_action` ***(String)****:* the name of the input action for negative direction **("ui_left")**
3. `positive_action` ***(String)****:* the name of the input action for positive direction **("ui_right")**

Returns ***(float):*** number between -1 and 1

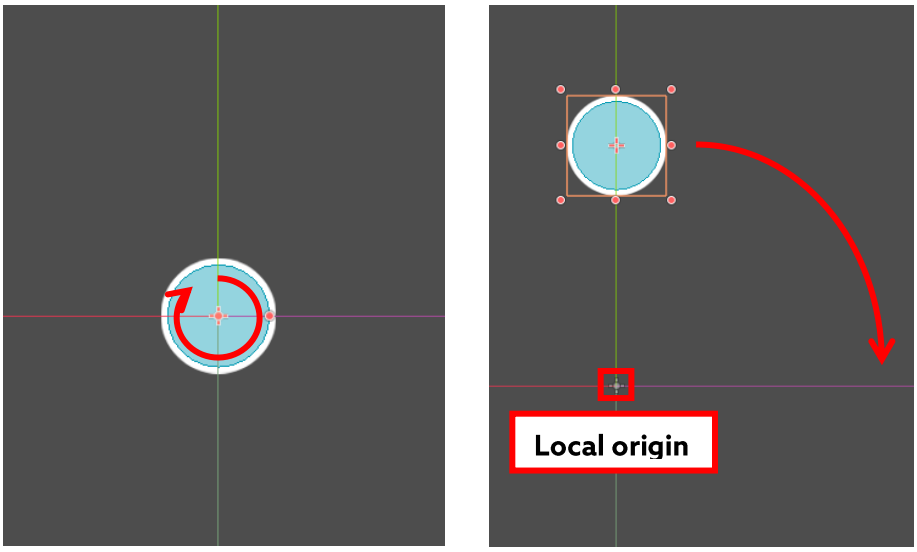
```
15    # Write the _process method
16    # ------------------
17    func _process(_delta):
18        movement = Input.get_axis("ui_left", "ui_right")
```

INSTANTIATE()

A method called on a **PackedScene** resource to create a live copy (an instance) of the scene's nodes, which can then be added to the scene tree.

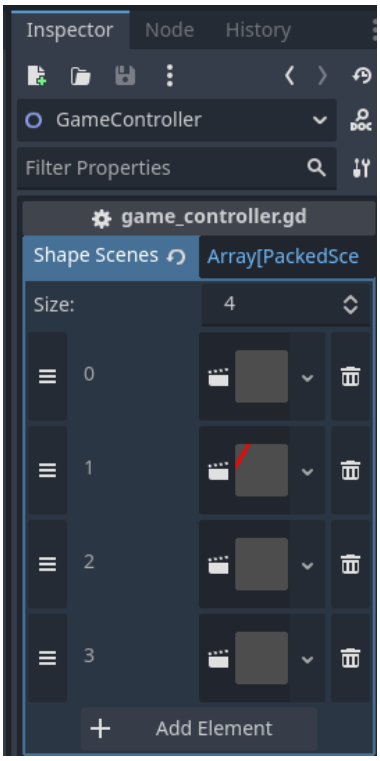
Local Origin

The original position that an object is relative to.



PackedScene

A resource that holds all the data from a saved scene. It does not exist in the **main scene tree** until it is **instanced** when the game is running. This makes it possible to load and spawn scenes dynamically such as creating enemies, obstacles, or other items when needed.



RANDI_RANGE()

Functions the same as the randint(min, max) function from MakeCode.

`randi_range()`: generates a random integer between the given minimum and maximum values, including both ends.

Parameters:

2. `from` ***(int)*:** the minimum value in the range
3. `to` ***(int)*:** the maximum value in the range

Returns *(int)*: A random number within the range.

ROTATE()

`rotate()`: part of the Node class, rotates a node by the given angle in radians around its local origin.

Parameters:

2. `radians` ***(float)*:** the angle to rotate in radians. Positive values rotate clockwise; negative values rotate counterclockwise.

Returns *(void)*: nothing

```
22    # Write the _physics_process method
23    # --------------------
24    func _physics_process(delta):
25        rotate(movement * delta * move_speed)
```

ACTIVITY 07: POLYRUN

IS_IN_GROUP()

`is_in_group()`: Part of the Node class, this method returns **true** if this node has been added to the given **group**.

Parameters:

1. `group` ***(String):*** the group to be checked (`"Player"`).

Returns *(Boolean):* true, if the node is found in the group.

```
7  func _on_body_entered(body):
8      if body.is_in_group("Player"):
9          Globals.game_end.emit()
```

IS_ON_WALL()

`is_on_wall()`: Part of the CharacterBody2D/3D classes, returns whether the body collided with a wall on the last call of **move_and_slide()**. The **up_direction** and **floor_max_angle** are used to determine whether a surface is "wall" or not.

Parameters: None

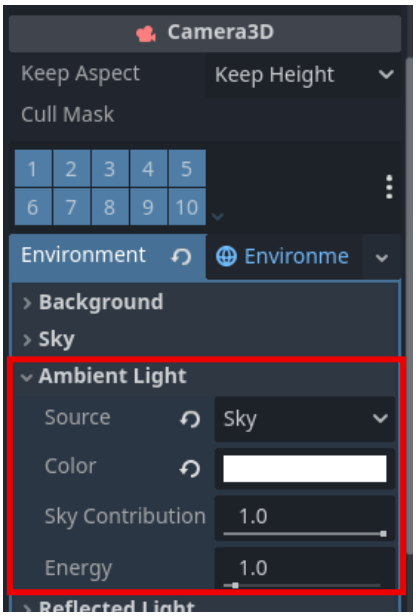
Returns *(Boolean):* whether the body last collided with a wall.

```
25
26      # --------------------
27      # TODO STEP 6
28      # Connect game_end & score_update signal
29      #
30      if is_on_wall():
31
32
```

Activity 08: Dropping Bombs Part 2

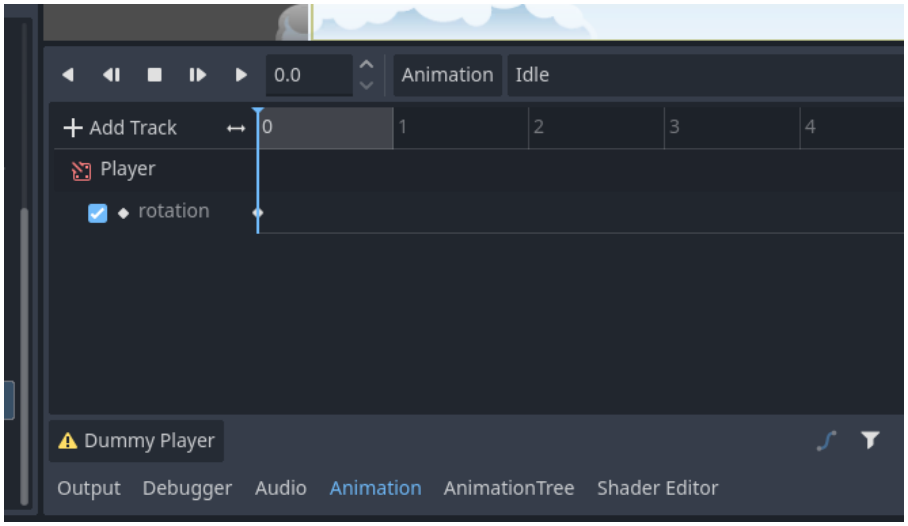
Ambient Light

A property of the Camera3D node that sets the color and strength of ambient light in the game from that camera.



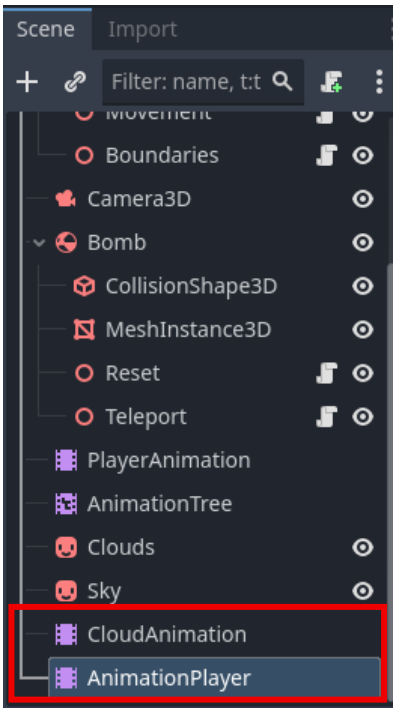
Animation Bottom Panel

An editor panel in Godot that allows the creation and editing of keyframes on a track.



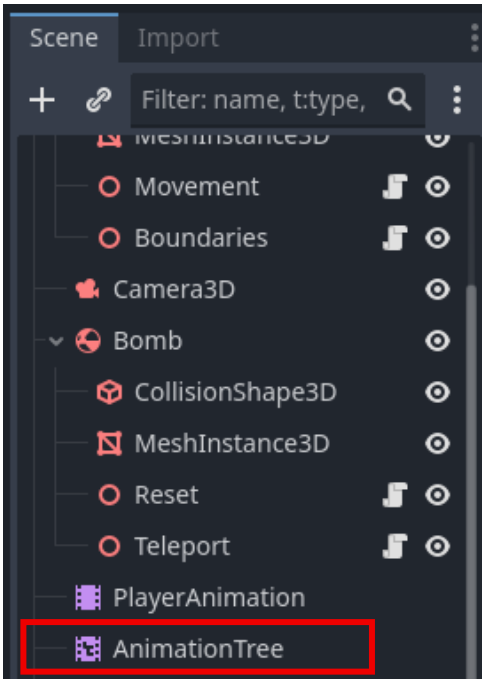
AnimationPlayer

A node that is used for general-purpose animation playback.



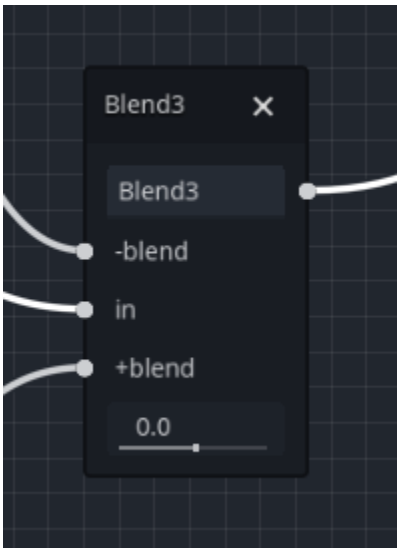
AnimationTree

A node used for advanced animation transitions in an **AnimationPlayer** node.



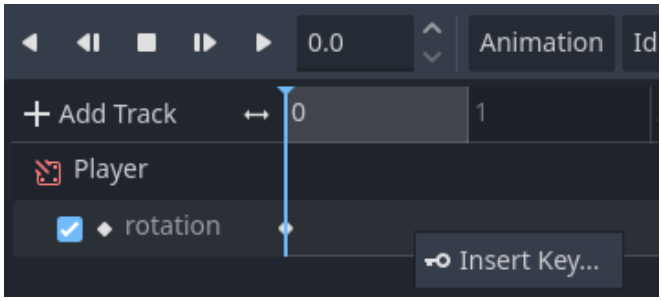
Blend3

An animation node that can be added in the Animation Track editor. It blends 3 input animation keyframes and includes an editable coefficient [-1, 1] that edits the blend amount of the resulting animation.



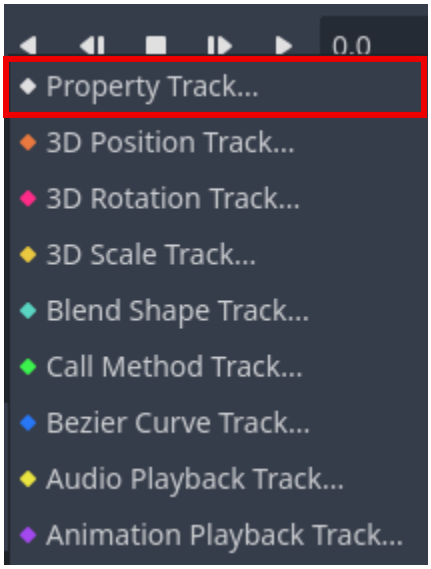
Key / Keyframe

A keyframe is essentially a "key" point of time in an animation where one of the properties are a certain value. Animations are transitions between keyframes.



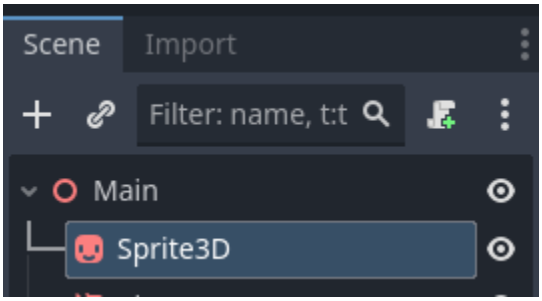
Property Track

A type of animation track that will edit one of the properties of a node.



Sprite3D

A node that allows a 2D texture to be displayed in a 3D environment.



Activity 09: Dropping Bombs Part 3

get_nodes_in_group()

`get_nodes_in_group()`: Part of the SceneTree class, this method returns an array of all nodes inside the tree which have been added to the given **group.**

Parameters:

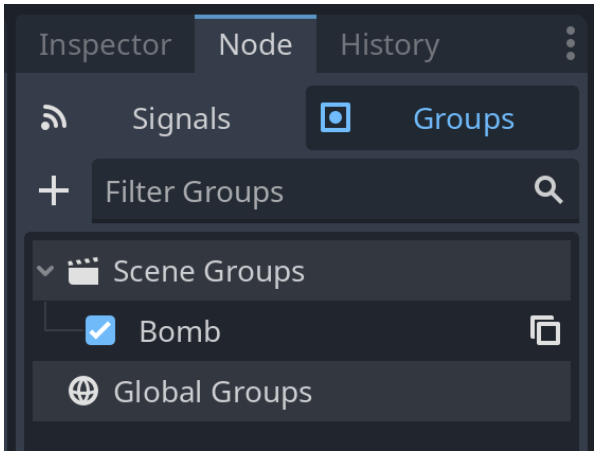
2. `group` ***(String):*** the group to be checked (**`"Player"`**).

Returns *(Array[Node]):* all nodes that are inside the tree with the given **group**.

```
30  # --------------------
31  # TODO 4
32  # Remove bomb objects when they've left the viewport
33  # --------------------
34  func _process(_delta: float) -> void:
35      var all_bombs = get_tree().get_nodes_in_group("Bomb")
```

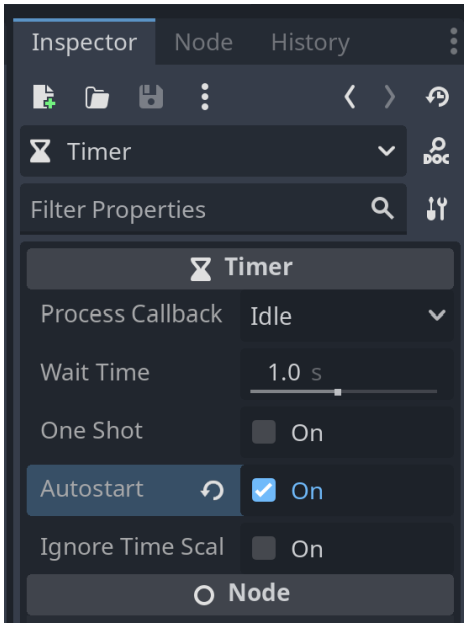
Groups

Groups act as Godot's tagging system, where the developer can assign multiple groups to each node. Groups allow the developer to perform actions (like calling methods) on all nodes in a group at once.



Timer Node

Remember how timers were used in Don't Touch the Cubes and Super Shapes? Well, those projects were really creating temporary Timer nodes! Example from Super Shapes: **await get_tree().create_timer(spawn_delay).timeout**



Viewport

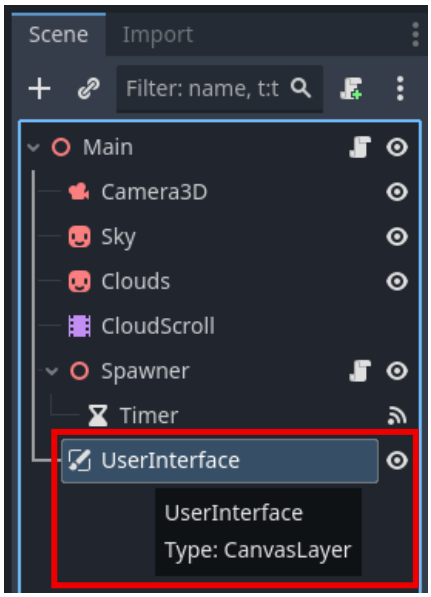
The **viewport** is a built-in feature of Godot. Think of it as the screen that Godot is drawing the game on, so the player can see it. The viewport will automatically associate itself with the highest Camera node in the **Scene** hierarchy.

```
12  # --------------------
13  # TODO 2
14  # Get the viewport size
15  # --------------------
16  func _ready() -> void:
17      spawn_x = get_viewport().size.x/100
18      spawn_y = get_viewport().size.y/100
19
```

Activity 10: Dropping Bombs Part 4

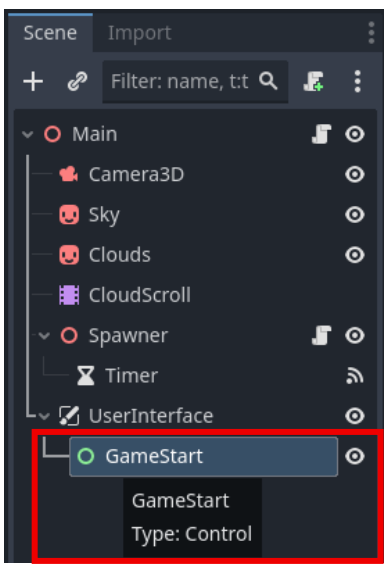
CanvasLayer

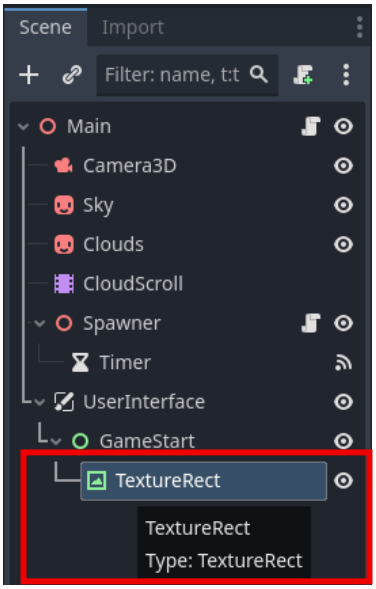
A node used to create UI. It creates a 2D overlay that supersedes all other layers in relation to the camera so that they'll appear at the front when the game runs.



Control Node

Nodes within CanvasLayer node that contain all the object nodes seen within UI, such as textures and labels.





TextureRect

A type of control node that has images in UI.

Activity 11: Dropping Bombs Part 5

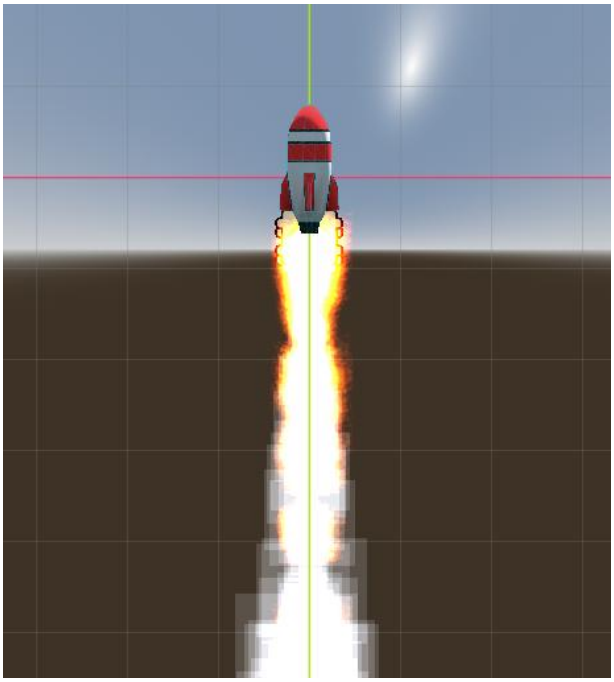
No new concepts

This activity focuses on refining the Dropping Bombs activity and does not introduce any new concepts.

Activity 12: Dropping Bombs Part 6

CPUParticles3D

A node that allows generation of advanced particles, providing many properties that may be tweaked.



EMITTING

A property of CPU/GPUParticles2D/3D - Determines whether new particles are being generated.

```
func _ready() -> void:
    contact_monitor = true
    max_contacts_reported = 1
    body_entered.connect(_on_body_entered)
    # TODO #1: turn of particles when bomb is created
    get_node("Explosion").emitting = false
```