

Godot Programming & Development

A Comprehensive Overview

What is Godot?

- Open-source game engine
- Scene-based architecture
- Node & tree hierarchy
- Multi-platform (PC, mobile, web, consoles)
- Built-in scripting (GDScript, C#, C++)
- Supports 2D, 3D, and XR development

Scene System

Everything is a node

- Scenes are trees of nodes
- Nodes inherit from base `Node` class
- Scenes can be instanced as nodes in other scenes
- Reusable, modular design

```
# Scene structure example
Root (Node2D)
└── Player (CharacterBody2D)
    ├── Sprite2D
    ├── CollisionShape2D
    └── Camera2D
└── Environment (Node2D)
```

GDScript Basics

Python-like syntax, optimized for Godot

```
extends Node2D

# Variables
var speed: float = 200.0
@export var health: int = 100

# Lifecycle methods
func _ready():
    print("Scene loaded")

func _process(delta):
    position.x += speed * delta

# Custom functions
func take_damage(amount: int):
    health -= amount
    if health <= 0:
        queue_free()
```

GDScript: Signals

Event-driven communication between nodes

```
# Define signal
signal health_changed(new_health)
signal died

# Emit signal
func take_damage(amount):
    health -= amount
    health_changed.emit(health)
    if health <= 0:
        died.emit()

# Connect signal
func _ready():
    player.health_changed.connect(_on_player_health_changed)

func _on_player_health_changed(new_health):
    print("Health: ", new_health)
```

2D Development - Core Nodes (1)

Node	Purpose
Node2D	Base 2D node with transform
Sprite2D	Display textures/images
AnimatedSprite2D	Frame-based sprite animation
characterBody2D	Physics body for characters
RigidBody2D	Physics-simulated body

2D Development - Core Nodes (2)

Node	Purpose
StaticBody2D	Non-moving collision body
Area2D	Detect overlaps, no physics
Camera2D	2D camera with smoothing
TileMap	Grid-based tile rendering

2D Example: Player Movement

```
extends CharacterBody2D

@export var speed = 300.0
@export var jump_velocity = -400.0

var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

func _physics_process(delta):
    # Gravity
    if not is_on_floor():
        velocity.y += gravity * delta

    # Jump
    if Input.is_action_just_pressed("jump") and is_on_floor():
        velocity.y = jump_velocity

    # Movement
    var direction = Input.get_axis("left", "right")
    velocity.x = direction * speed

move_and_slide()
```

3D Development - Core Nodes (1)

Node	Purpose
Node3D	Base 3D node with 3D transform
MeshInstance3D	Display 3D meshes
CharacterBody3D	Physics body for 3D characters
RigidBody3D	Physics-simulated 3D body
StaticBody3D	Static 3D collision body

3D Development - Core Nodes (2)

Node	Purpose
Area3D	3D overlap detection
Camera3D	3D camera with projection
DirectionalLight3D	Sun-like lighting
OmniLight3D	Point light source
SpotLight3D	Spotlight with cone

3D Example: FPS Controller (Setup)

```
extends CharacterBody3D

@export var speed = 5.0
@export var jump_velocity = 4.5
@export var mouse_sensitivity = 0.002

@onready var camera = $Camera3D
var gravity = 9.8

func _ready():
    Input.mouse_mode = Input.MOUSE_MODE_CAPTURED

func _unhandled_input(event):
    if event is InputEventMouseMotion:
        rotate_y(-event.relative.x * mouse_sensitivity)
        camera.rotate_x(-event.relative.y * mouse_sensitivity)
        camera.rotation.x = clamp(camera.rotation.x, -PI/2, PI/2)
```

3D Example: FPS Controller (Movement)

```
func _physics_process(delta):
    # Apply gravity
    if not is_on_floor():
        velocity.y -= gravity * delta

    # Jump
    if Input.is_action_just_pressed("jump") and is_on_floor():
        velocity.y = jump_velocity

    # Movement input
    var input_dir = Input.get_vector("left", "right", "forward", "back")
    var direction = (transform.basis * Vector3(input_dir.x, 0, input_dir.y)).normalized()

    velocity.x = direction.x * speed
    velocity.z = direction.z * speed

move_and_slide()
```

XR Development

OpenXR Support for VR/AR

Key XR Nodes & Concepts:

Node	Purpose
XROrigin3D	XR coordinate system origin
XRCamera3D	HMD-tracked camera
XRController3D	VR controller tracking
XRNode3D	Generic tracked node

- Hand tracking
- Controller input
- Passthrough (AR)
- 6DOF tracking

Timers

Timer Node - Execute code after delay

```
extends Node

@onready var timer = $Timer

func _ready():
    # One-shot timer
    timer.wait_time = 2.0
    timer.one_shot = true
    timer.timeout.connect(_on_timer_timeout)
    timer.start()

    # Or create programmatically
    var new_timer = Timer.new()
    add_child(new_timer)
    new_timer.wait_time = 1.0
    new_timer.autostart = true
    new_timer.timeout.connect(_on_repeat_timer)

func _on_timer_timeout():
    print("Timer finished!")

func _on_repeat_timer():
    print("Every second")
```

Tweens

Animate properties over time

```
extends Sprite2D

func _ready():
    # Fade in
    modulate.a = 0
    var tween = create_tween()
    tween.tween_property(self, "modulate:a", 1.0, 1.5)

    # Chain animations
    tween = create_tween()
    tween.tween_property(self, "position:x", 500, 2.0)
    tween.tween_property(self, "rotation", PI, 1.0)
    tween.tween_callback(animation_complete)

    # Parallel tweens
    tween = create_tween().set_parallel(true)
    tween.tween_property(self, "scale", Vector2(2, 2), 1.0)
    tween.tween_property(self, "modulate", Color.RED, 1.0)

func animation_complete():
    print("Animation done!")
```

Tween Options

```
func complex_tween():
    var tween = create_tween()

    # Easing and transitions
    tween.set_ease(Tween.EASE_IN_OUT)
    tween.set_trans(Tween.TRANS_ELASTIC)

    # Duration-based
    tween.tween_property(self, "position", Vector2(100, 100), 2.0)

    # Relative movement
    tween.tween_property(self, "position:x", 50, 1.0).as_relative()

    # Looping
    tween.set_loops(5)    # Loop 5 times
    tween.set_loops()     # Infinite loop

    # Delays and intervals
    tween.tween_interval(1.0) # Wait 1 second

    # Animate method calls
    tween.tween_method(set_health, 0, 100, 2.0)
```

AnimationPlayer

Timeline-based animation system

```
extends Node2D

@onready var anim_player = $AnimationPlayer

func _ready():
    # Play animation
    anim_player.play("idle")

    # Connect signals
    anim_player.animation_finished.connect(_on_animation_finished)

    # Control playback
    anim_player.speed_scale = 2.0 # Play at 2x speed

func attack():
    anim_player.play("attack")

func _on_animation_finished(anim_name):
    if anim_name == "attack":
        anim_player.play("idle")
```

AnimationPlayer: Creating Animations

In Godot Editor:

1. Add AnimationPlayer node
2. Create new animation
3. Select properties to animate (position, rotation, sprite frame, etc.)
4. Add keyframes at different times
5. Adjust easing curves

Key Features:

- Multiple tracks (property, method call, audio)
- Bezier curve editing
- Animation blending
- Call methods at specific times

Animation Example: Character States (1)

```
extends CharacterBody2D

@onready var anim_player = $AnimationPlayer
@onready var sprite = $Sprite2D

enum State { IDLE, WALK, JUMP, ATTACK }
var current_state = State.IDLE

func _physics_process(delta):
    match current_state:
        State.IDLE:
            anim_player.play("idle")
            if velocity.x != 0:
                change_state(State.WALK)
        State.WALK:
            anim_player.play("walk")
            if velocity.x == 0:
                change_state(State.IDLE)
```

Animation Example: Character States (2)

```
func _physics_process(delta):
    # ... continued
    match current_state:
        State.JUMP:
            anim_player.play("jump")
            if is_on_floor():
                change_state(State.IDLE)

    # Flip sprite based on direction
    if velocity.x < 0:
        sprite.flip_h = true
    elif velocity.x > 0:
        sprite.flip_h = false

func change_state(new_state):
    current_state = new_state
```

AnimationTree

Advanced animation blending

```
extends CharacterBody3D

@onready var anim_tree = $AnimationTree
@onready var playback = anim_tree.get("parameters/playback")

func _physics_process(delta):
    # Blend between idle and walk based on speed
    var speed_ratio = velocity.length() / max_speed
    anim_tree.set("parameters/idle_walk_blendblend_amount", speed_ratio)

    # State machine transitions
    if Input.is_action_just_pressed("attack"):
        playback.travel("attack")
    elif is_on_floor():
        playback.travel("ground_movement")
    else:
        playback.travel("in_air")
```

Use Cases:

- Smooth locomotion blending

Common Node Utilities

AudioStreamPlayer - Play sound effects/music

```
$AudioStreamPlayer.play()  
$AudioStreamPlayer.stream = preload("res://sound.ogg")
```

CanvasLayer - UI layering

```
var ui_layer = CanvasLayer.new()  
ui_layer.layer = 10 # Draw on top
```

HTTPRequest - Web requests

```
func _ready():  
    $HTTPRequest.request_completed.connect(_on_request)  
    $HTTPRequest.request("https://api.example.com/data")
```

FileAccess - Save/load data

```
var file = FileAccess.open("user://save.dat", FileAccess.WRITE)
```

Input Handling

```
func _input(event):
    # Keyboard
    if event.is_action_pressed("ui_accept"):
        jump()

    # Mouse
    if event is InputEventMouseButton:
        if event.button_index == MOUSE_BUTTON_LEFT and event.pressed:
            shoot()

    # Touch (mobile)
    if event is InputEventScreenTouch:
        if event.pressed:
            touch_position = event.position

func _process(delta):
    # Continuous input
    var direction = Input.get_vector("left", "right", "up", "down")
    velocity = direction * speed
```

Input Map (Project Settings → Input Map):

- Define actions ("jump", "attack", etc.)
- Map to keys, buttons, axes

Signals & Groups

Built-in Signals:

```
# Button
button.pressed.connect(_on_button_pressed)

# Area2D
area.body_entered.connect(_on_body_entered)

# Timer
timer.timeout.connect(_on_timeout)
```

Groups:

```
# Add to group
add_to_group("enemies")

# Check membership
if is_in_group("enemies"):
    take_damage()

# Call method on all group members
get_tree().call_group("enemies", "alert")

# Get all nodes in group
var_all_enemies = get_tree().get_nodes_in_group("enemies")
```

Resource System

Reusable data containers

```
# Define resource (weapon_data.gd)
class_name WeaponData
extends Resource

@export var weapon_name: String
@export var damage: int
@export var fire_rate: float
@export var icon: Texture2D

# Use resource
extends Node

@export var weapon: WeaponData

func _ready():
    print(weapon.weapon_name)
    print("Damage: ", weapon.damage)
```

Create in Editor:

- Right-click → New Resource

Autoload (Singletons)

Global scripts accessible everywhere

1. Create script (e.g., `game_manager.gd`)
2. Project Settings → Autoload → Add script
3. Access from any scene

```
# game_manager.gd
extends Node

var score: int = 0
var player_health: int = 100

signal score_changed(new_score)

func add_score(amount: int):
    score += amount
    score_changed.emit(score)

# Access from anywhere
func _ready():
    GameManager.add_score(10)
    print(GameManager.score)
```

Debugging Tips

Print Debugging:

```
print("Value: ", variable)  
print_debug("With stack trace")
```

Breakpoints:

- Click line number gutter in script editor
- Pause execution
- Inspect variables

Remote Scene Tree:

- Debug → Remote in editor
- Inspect running game's node tree

Performance Monitor:

Project Organization

```
res://  
└── scenes/  
    ├── characters/  
    ├── levels/  
    └── ui/  
└── scripts/  
    ├── autoload/  
    └── resources/  
└── assets/  
    ├── sprites/  
    ├── models/  
    ├── audio/  
    └── fonts/  
└── addons/
```

Export & Deployment

Export Templates:

- Download from Editor → Manage Export Templates
- Or use one-click deploy

Performance Optimization

General Tips:

- Use signals instead of polling
- Defer expensive operations to separate threads
- Cache node references in `_ready()`

2D Specific:

- Use TileMap for repetitive backgrounds

3D Specific:

- LOD (Level of Detail) meshes
- Occlusion culling
- Lightmap baking for static lighting
- Instance MultiMeshInstance3D for lots of objects

Useful Resources

Official:

- docs.godotengine.org
- godotengine.org/community
- github.com/godotengine/godot

Learning:

- GDQuest (YouTube/courses)
- Brackeys Godot tutorials
- HeartBeast game dev
- Official demos (built-in Asset Library)

Community:

- [/r/godot](https://www.reddit.com/r/godot)

Key Takeaways

- Scene-based architecture** - Everything is a node in a tree
- GDScript is Python-like** - Easy to learn, powerful
- Physics bodies** - Character, Rigid, Static, Area
- Timers** - Delayed execution
- Tweens** - Smooth property interpolation
- AnimationPlayer** - Timeline-based animations
- Signals** - Event-driven communication
- 2D, 3D, XR** - One engine, multiple dimensions

Start small, build up, iterate!