Please generate for me a 2D godot physics character controller

it should have:

A modular movement system such that I can add movement behaviors without having to change existing movements (eg, adding run, dash, roll etc)

A modular and blended animation system such that I can add animations at a later date without impacting existing animations.

Below is a **modular 2D CharacterBody2D controller architecture for Godot 4** designed around:

- ✅ **Pluggable movement behaviors** (run, dash, roll, etc. added as separate scripts)
- ✅ **Modular animation system** with blending via `AnimationTree`
- ✅ No need to modify existing behaviors when adding new ones
- ✅ Clean separation of responsibilities

This is built using:

- `CharacterBody2D`
- `Resource` -based movement modules
- A movement stack / priority system
- `AnimationTree` with a BlendSpace2D + state machine
- Signal-based communication

🧱 Scene Structure

scss Copy code

```
Player (CharacterBody2D)
├── AnimationPlayer
├── AnimationTree
├── Sprite2D
├── MovementManager (Node)
└── AnimationController (Node)
```

🎮 1. Core Player Script

Attach to `Player (CharacterBody2D)`

gdscript Copy code

```
extends CharacterBody2D

@export var gravity: float = 1200.0

@onready var movement_manager = $MovementManager
@onready var animation_controller = $AnimationController

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

    movement_manager.process_movement(self, delta)

    move_and_slide()

    animation_controller.update_animation(self)
```

🧩 2. Modular Movement System

The key idea:

Each movement behavior is a `Resource` that can:

- Decide if it wants control
- Modify velocity
- Have priority

The MovementManager picks the highest priority active movement.

🧠 Base Movement Behavior (Resource)

Create: `movement_behavior.gd`

gdscript Copy code

```
extends Resource
class_name MovementBehavior

@export var priority: int = 0

func wants_control(player: CharacterBody2D) -> bool:
    return false

func process(player: CharacterBody2D, delta: float) -> void:
    pass
```

🎛️ Movement Manager

Attach to `MovementManager` node

gdscript Copy code

```
extends Node

@export var behaviors: Array[MovementBehavior]

var active_behavior: MovementBehavior

func process_movement(player: CharacterBody2D, delta: float):
```

```
    var highest_priority := -999
    var chosen_behavior: MovementBehavior = null

    for behavior in behaviors:
        if behavior.wants_control(player):
            if behavior.priority > highest_priority:
                highest_priority = behavior.priority
                chosen_behavior = behavior

    if chosen_behavior:
        active_behavior = chosen_behavior
        chosen_behavior.process(player, delta)
```

✅ Add new movement types by:

- Creating a new `Resource`
- Adding it to the `behaviors` array
- No changes required elsewhere

🚶 Example Movement: Walk / Run

Create: `move_walk.gd`

gdscript Copy code

```
extends MovementBehavior
class_name MoveWalk

@export var speed: float = 200.0
@export var acceleration: float = 800.0

func wants_control(player):
    return Input.get_axis("move_left", "move_right") != 0

func process(player, delta):
    var direction = Input.get_axis("move_left", "move_right")
    player.velocity.x = move_toward(
        player.velocity.x,
        direction * speed,
```

```
            acceleration * delta
        )
```

⚡ Example Movement: Dash

Create: `move_dash.gd`

gdscript Copy code

```
extends MovementBehavior
class_name MoveDash

@export var dash_speed: float = 600.0
@export var dash_time: float = 0.2
@export var cooldown: float = 0.5
@export var priority := 10

var dash_timer := 0.0
var cooldown_timer := 0.0
var direction := 0

func wants_control(player):
    if cooldown_timer > 0:
        return false
    return Input.is_action_just_pressed("dash")

func process(player, delta):
    if dash_timer <= 0:
        direction = sign(player.velocity.x)
        if direction == 0:
            direction = 1
        dash_timer = dash_time
        cooldown_timer = cooldown

    dash_timer -= delta
    cooldown_timer -= delta

    player.velocity.x = direction * dash_speed
```

Dash overrides walk automatically due to higher priority.

🎞️ Modular Animation System

We separate:

- Movement logic
- Animation logic

AnimationController reads player state and updates AnimationTree.

🎬 AnimationTree Setup

Use:

- AnimationNodeStateMachine
- BlendSpace2D for locomotion
- Separate Dash state
- Separate Roll state
- Idle state

Your tree could look like:

scss Copy code

```
StateMachine
 ├── Locomotion (BlendSpace2D)
 ├── Dash
 ├── Roll
 └── Jump
```

🎛️ Animation Controller

Attach to `AnimationController`

gdscript Copy code

```
extends Node
```

```
@onready var tree: AnimationTree = $"../AnimationTree"
@onready var state_machine = tree.get("parameters/playback")

func _ready():
    tree.active = true

func update_animation(player: CharacterBody2D):

    var speed = abs(player.velocity.x)

    if not player.is_on_floor():
        state_machine.travel("Jump")
        return

    if speed < 10:
        state_machine.travel("Idle")
    else:
        state_machine.travel("Locomotion")

    tree.set("parameters/Locomotion/blend_position", Vector2(speed, 0))
```

🧩 Adding New Animations Later

To add:

Example: Add Roll

1. Add Roll animation to AnimationPlayer
2. Add Roll state in AnimationTree
3. Add MoveRoll behavior
4. In AnimationController:

gdscript Copy code

```
if player.movement_manager.active_behavior is MoveRoll:
    state_machine.travel("Roll")
```

Done.

No existing animation logic changed.

🧠 Why This Architecture Works

Movement

- Behaviors are isolated
- Priority-based override
- No hardcoded state machine
- Add/remove without modification

Animation

- Fully decoupled from movement
- Driven by state + velocity
- Uses AnimationTree blending
- Easy to extend

🚀 How To Add A New Movement Later

1. Create new MovementBehavior resource
2. Set priority
3. Add to MovementManager.behaviors
4. Optionally add animation state

No refactoring required.

🔥 Optional Advanced Improvements

If you want this production-grade:

- Convert MovementBehavior to Nodes instead of Resources (for internal timers)
- Use signals for animation state changes
- Add interrupt rules between behaviors
- Add buffered input system
- Add coyote time & jump buffering module
- Use a MovementContext struct for cleaner data passing

If you'd like, I can now provide:

- 🔁 A version using Node-based behaviors (better for complex moves)
- 🎮 A full platformer-ready controller (jump, coyote, wall slide)
- ⚙️ A fully data-driven state machine version
- 🧱 An ECS-style architecture
- 🎯 A combat-ready version with attack cancel windows

Just tell me what kind of game you're building.

Please give a prompt for this conversation

Here is a clean, high-quality prompt you can reuse to generate this kind of system:

Prompt:

Generate a production-ready 2D character controller for Godot 4 using `CharacterBody2D`.

Requirements:

1. **"The movement system must be modular and extensible."**
   - **"Movement behaviors (walk, run, dash, roll, jump, etc.) must be addable without modifying existing movement code."**
   - **"Use a clean architecture (composition over inheritance)."**
   - **"Support priorities or a behavior selection system."**
   - **"Avoid tight coupling between movement behaviors."**
2. **"The animation system must also be modular and scalable."**
   - **"Use `AnimationTree`."**
   - **"Allow adding new animations later without modifying existing animation logic."**
   - **"Support blending (e.g., BlendSpace2D for locomotion)."**
   - **"Clean separation between movement logic and animation logic."**
3. **"Provide:"**

- **“Scene structure”**
- **“Base movement behavior implementation”**
- **“Movement manager”**
- **“Example behaviors (walk + dash)”**
- **“Animation controller”**
- **“Explanation of architecture decisions”**

4. **“Follow good software architecture principles (SRP, OCP, modularity, extensibility).”**

Output complete GDScript examples compatible with Godot 4.

If you'd like, I can also provide:

- A **shorter minimal prompt**
- A **more advanced architecture prompt (AAA-scale ready)**
- A **prompt optimized for AI coding assistants**
- A **prompt for a node-based ability system instead**
- A **prompt tailored for platformers vs top-down games**

Just tell me your use case.