Lesson 4 Quick Reference Guide

Adding Enemies & Basic AI to Your Platformer

Duration: 40 minutes
Godot Version: 4.5
Prerequisites: Completed Lessons 1-3

🎯 Learning Intentions

By the end of this lesson, you will:

- Understand how to create timer-based patrol AI behavior
- Learn about collision layers and masks
- Implement danger/consequence mechanics
- Create reusable enemy scenes
- Use timers to control movement patterns
- Understand the importance of call_deferred() for scene reloading

---

✅ Success Criteria

You will be successful when you can:

- [ ] Create an Enemy scene with proper collision setup
- [ ] Implement timer-based back-and-forth patrol movement
- [ ] Configure collision layers so enemies don't collide with each other
- [ ] Make the player restart level when touching enemy
- [ ] Use call_deferred() to safely reload scenes
- [ ] Explain the difference between collision layers and masks
- [ ] Test and debug AI behavior
- [ ] Adjust patrol timing and speed for different enemy types

---

📚 Key Vocabulary

- **AI (Artificial Intelligence):** Code that makes game objects behave on their own
- **Patrol:** Moving back and forth between two points
- **Timer:** Counting seconds to control when actions happen
- **Collision Layer:** Which layer this object is ON
- **Collision Mask:** Which layers this object can DETECT
- **State:** Current condition/behavior of an object (moving left, moving right)
- **Direction:** Which way something is facing/moving (-1 = left, 1 = right)
- **call_deferred():** Safely delays an action until physics is finished
- **Delta:** Time elapsed since the last frame (in seconds)

🎮 What We're Building

An enemy that:

- Moves left and right automatically using a timer
- Turns around every few seconds
- Restarts the level when player touches it
- Doesn't collide with other enemies
- Uses simple, reliable patrol behavior

Movement Pattern:

Time: 0s → 2s 2s → 4s 4s → 6s

Direction: RIGHT → LEFT → RIGHT

------→ ←------ ------→

📁 Project Structure After Lesson 4

Year9_Platformer/

├─ level_1.tscn ← Main gameplay scene

├─ level_2.tscn ← Victory screen scene

├─ player.gd ← Player movement script

├─ enemy.tscn ← NEW: Reusable enemy scene

└─ enemy.gd ← NEW: Enemy AI script

🔧 **Step-by-Step Instructions**

1. Create Enemy Scene (5 min)

A. Create New Scene

1. Click **Scene** menu → **New Scene** (or press Ctrl+N)
2. Select **Other Node**
3. Search for and create: **CharacterBody2D**
4. Rename it to: **Enemy**
5. Save scene as: **enemy.tscn** (Ctrl+S)

B. Add Collision Shape

1. Select **Enemy** node
2. Click **+** to add child node
3. Add: **CollisionShape2D**
4. In Inspector, click **Shape** → **New RectangleShape2D**
5. Use **Scale tool** to resize: approximately **32×32 pixels**

C. Add Visual (ColorRect)

1. Select **Enemy** node
2. Add child: **ColorRect**
3. In Inspector:
    - **Size:** 32×32
    - **Position:** -16, -16 (centers it)
    - **Color:** Red (255, 0, 0)

D. Add HitBox for Player Detection

1. Select **Enemy** node
2. Add child: **Area2D**
3. Rename it to: **HitBox**

4. Select **HitBox**
5. Add child: **CollisionShape2D**
6. Set **Shape: New RectangleShape2D**
7. Resize to match enemy size (32×32)

Your node tree should look like:

Enemy (CharacterBody2D)

├─ CollisionShape2D ← For physics (walls/platforms)

├─ ColorRect ← Visual appearance

└─ HitBox (Area2D) ← For detecting player

└─ CollisionShape2D ← Detection zone

💾 **Save:** Press Ctrl+S

---

2. Understanding Collision Layers (5 min)

What Are Collision Layers?

Think of collision layers like floors in a building:

- **Layer 1:** Player's floor
- **Layer 2:** Enemy's floor
- **Layer 3:** Coin's floor
- **Layer 4:** Environment's floor (platforms, walls)

Layer vs Mask

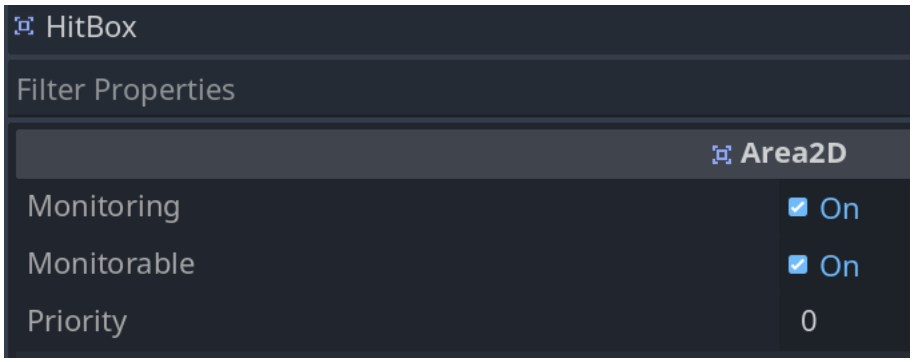
Property	Meaning	Example
Collision Layer	Which floor am I ON?	Enemy is on Layer 2
Collision Mask	Which floors can I DETECT?	Enemy can detect Layer 1 & 4

Set Up Enemy Collision

1. Select **Enemy** (CharacterBody2D) node
2. In Inspector, find **Collision** section
3. **Collision Layer:**
    - Click the grid
    - Enable **ONLY Layer 2** (second box)
4. **Collision Mask:**
    - Enable **Layer 4** (Platforms/walls)
    - Disable Layer 2 (enemies don't collide with each other!)

Set Up HitBox Collision

1. Select **HitBox** (Area2D) node
2. In Inspector, find **Collision** section
3. **Monitoring:** ✓ (checked)
4. **Monitorable:** ✓ (checked)



5. **Collision Layer:** Enable **ONLY Layer 2**
6. **Collision Mask:** Enable **ONLY Layer 1** (Player)

Verify Player Collision

1. Open **level_1.tscn**
2. Select **Player** node (might be named PlayerL2 or similar)
3. **Collision Layer:** Layer 1 should be checked ✓
4. **Collision Mask:** Should include Layer 2 and Layer 4

Verify Platform Collision

1. Select **Platform** node (or Platforms2, etc.)
2. **Collision Layer:** Enable **ONLY Layer 4**
3. **Collision Mask:** Can leave empty

💾 **Save both scenes** (Ctrl+Shift+S)

---

3. Create Enemy AI Script (10 min)

1. Open **enemy.tscn**
2. Select **Enemy** root node
3. Click **Attach Script** button (scroll icon)
4. Keep default settings

5. Click **Create**

Replace ALL code with:

```
extends CharacterBody2D

# === MOVEMENT CONSTANTS ===
# How fast the enemy moves (pixels per second)
const SPEED = 100.0

# === MOVEMENT VARIABLES ===
# Current direction: 1 = moving right, -1 = moving left
var direction = 1

# Timer to track when to turn around
var move_timer = 0.0

# How long to move in each direction (in seconds)
const MOVE_TIME = 2.0

# === INITIALIZATION ===
# Called once when enemy is created
func _ready():
	print("=== ENEMY SPAWNED ===")

	# Connect HitBox signal if it exists
	if has_node("HitBox"):
		print("✓ HitBox found and connected!")
		# Connect the signal to detect player collision
		$HitBox.body_entered.connect(_on_hit_box_body_entered)
```

```
        else:
            print("✗ ERROR: HitBox not found!")

# === MAIN PHYSICS FUNCTION ===
# Called every frame (delta = time since last frame)
func _physics_process(delta):
        # --- APPLY GRAVITY ---
        # Make enemy fall if not on ground
        if not is_on_floor():
            velocity += get_gravity() * delta

        # --- MOVE IN CURRENT DIRECTION ---
        # Multiply direction (1 or -1) by SPEED to get velocity
        velocity.x = direction * SPEED

        # --- COUNT UP THE TIMER ---
        # Add elapsed time to the timer
        move_timer += delta

        # --- CHECK IF TIME TO TURN AROUND ---
        if move_timer >= MOVE_TIME:
            # Time's up! Reverse direction
            direction *= -1  # Flip: 1 becomes -1, -1 becomes 1
            move_timer = 0.0  # Reset timer back to zero
            print("Enemy turned! Now moving: ", "RIGHT" if direction == 1 else "LEFT")

        # --- EXECUTE THE MOVEMENT ---
        # Actually move the enemy based on velocity
```

```
        move_and_slide()


# === PLAYER COLLISION ===
# Called when something enters the HitBox
func _on_hit_box_body_entered(body):
        print("Something hit the enemy: ", body.name)


        # Check if "Player" is anywhere in the body's name
        # This works for: Player, Player2, PlayerL2, etc.
        if "Player" in body.name:
                print("💀 Player hit enemy! Restarting level...")
                # Use call_deferred to safely reload after physics finishes
                get_tree().call_deferred("reload_current_scene")
```

💾 **Save:** Ctrl+S

🎓 **Understanding the Code**

How the Timer Works

move_timer += delta # Add time each frame

- **Frame 1:** move_timer = 0.016 (60 FPS)
- **Frame 2:** move_timer = 0.032
- **...keeps adding...**
- **Frame 120:** move_timer = 2.0 → TURN!

Direction Multiplication

direction *= -1

This flips the direction:

- If direction = 1: 1 × -1 = -1 (right becomes left)
- If direction = -1: -1 × -1 = 1 (left becomes right)

Why call_deferred()?

get_tree().call_deferred("reload_current_scene")

Without call_deferred:

- Tries to reload immediately during physics calculation
- Causes errors: "Removing a CollisionObject during physics callback"

With call_deferred:

- Waits for physics to finish
- Then safely reloads the scene
- No errors!

Analogy: It's like waiting for someone to finish their sentence before interrupting, instead of cutting them off mid-word.

---

4. Add Enemy to Level (3 min)

1. Open **level_1.tscn**
2. Click **Instantiate Child Scene** button (chain link icon at top left)
3. Select **enemy.tscn**
4. **Position enemy** on your platform using **Move tool**
5. **Test position:** Enemy should be on the platform, not floating

Add Multiple Enemies (Optional)

1. Select the enemy you just added
2. Press **Ctrl+D** to duplicate
3. Move duplicate to a different location
4. Repeat for more enemies

💾 **Save:** Ctrl+S

---

5. Test Your Enemy! (3 min)

Testing Checklist:

1. Press **F5** to run
2. **Watch Output panel** - should see:
3. === ENEMY SPAWNED ===✓ HitBox found and connected!
4. Verify enemy behavior:
    - [ ] Enemy moves smoothly
    - [ ] Enemy changes direction every 2 seconds
    - [ ] See "Enemy turned!" messages in Output
    - [ ] Enemy stays on platform (doesn't float or fall through)
5. **Test collision:**
    - [ ] Walk into enemy
    - [ ] Level restarts
    - [ ] See "Player hit enemy!" message

If Multiple Enemies:

- [ ] Enemies pass through each other (don't collide)
- [ ] Each enemy turns independently
- [ ] All enemies detect player

🎨 **Adjusting Enemy Behavior**

Make Enemy Move Longer Before Turning

```
const MOVE_TIME = 3.0  # 3 seconds each direction
```

Make Enemy Move Shorter Before Turning

```
const MOVE_TIME = 1.0  # 1 second each direction
```

Make Enemy Faster

```
const SPEED = 150.0  # Faster!
```

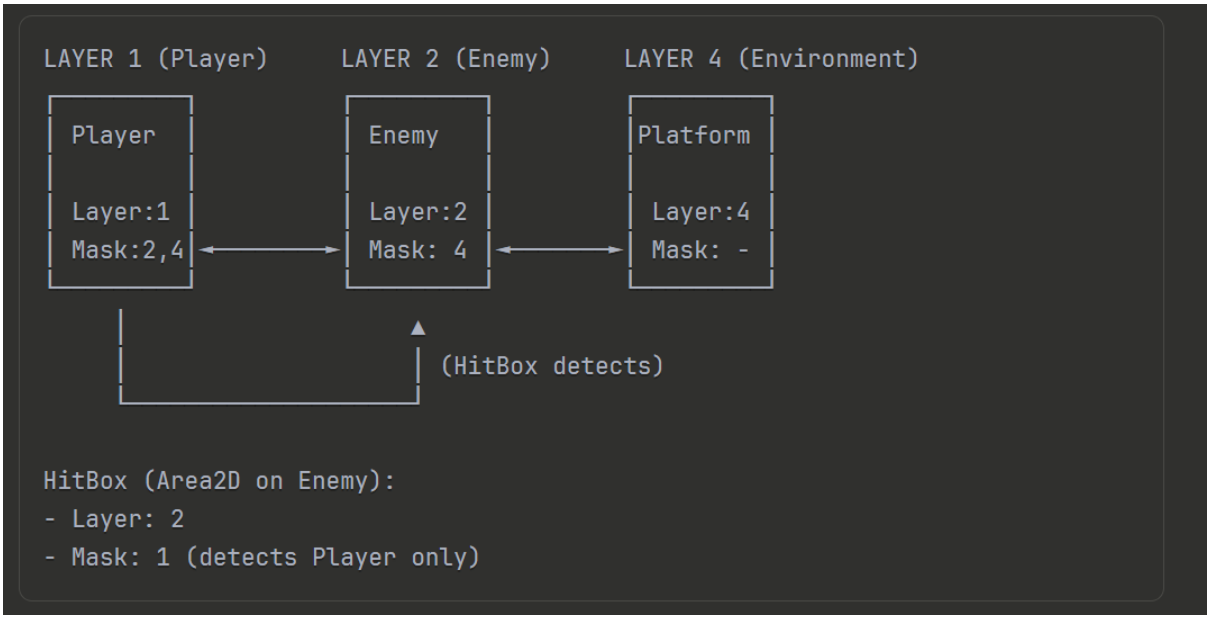
Make Enemy Slower

```
const SPEED = 50.0  # Slower!
```

Student Activity: Find Perfect Balance

MOVE_TIME	SPEED	Difficulty	Fun Rating (1-5)
2.0	100	Default	?
1.0	100	Quick turns	?
3.0	100	Long patrol	?
2.0	150	Fast enemy	?
2.0	50	Slow enemy	?

🧠 Collision Layers Diagram



Layer Setup Summary:

Object	Layer (What I am)	Mask (What I detect)
Player (CharacterBody2D)	1	2 (Enemy), 4 (Platform)
Enemy (CharacterBody2D)	2	4 (Platform only)
HitBox (Area2D)	2	1 (Player only)
Platform (StaticBody2D)	4	None needed

Key Point: The Enemy's CharacterBody2D and HitBox have different masks!

- **CharacterBody2D** mask = Layer 4 (so enemy collides with platforms)
- **HitBox** mask = Layer 1 (so HitBox detects player only)

🏆 Challenge Activities

Challenge 1: Different Enemy Types

Create 3 different enemy scenes with different speeds:

Red Enemy (Normal):

const SPEED = 100.0

const MOVE_TIME = 2.0

Blue Enemy (Slow & Cautious):

const SPEED = 50.0

const MOVE_TIME = 3.0

Yellow Enemy (Fast & Aggressive):

const SPEED = 150.0

const MOVE_TIME = 1.0

Change the ColorRect color to match!

Challenge 2: Visual Direction Indicator

Make the enemy change color when it turns:

```
func _physics_process(delta):
    # ... existing code ...

    # Change color based on direction
    if has_node("ColorRect"):
        if direction == 1:
            $ColorRect.color = Color.RED  # Moving right
        else:
            $ColorRect.color = Color.DARK_RED  # Moving left
```

Challenge 3: Distance-Based Patrol

Instead of time, patrol a fixed distance:

```
# Add at top with other variables
var start_position = 0.0
const PATROL_DISTANCE = 200.0  # Pixels to move each way

func _ready():
	# Remember starting position
	start_position = global_position.x

	if has_node("HitBox"):
		$HitBox.body_entered.connect(_on_hit_box_body_entered)

func _physics_process(delta):
	if not is_on_floor():
		velocity += get_gravity() * delta

	velocity.x = direction * SPEED

	# Check distance traveled from start
	var distance_traveled = abs(global_position.x - start_position)

	# Turn around if moved far enough
	if distance_traveled >= PATROL_DISTANCE:
		direction *= -1
		start_position = global_position.x  # New start point
		print("Enemy turned after traveling ", PATROL_DISTANCE, " pixels!")

	move_and_slide()
```

Challenge 4: Patrol Between Markers

Use Marker2D nodes to set exact patrol boundaries:

1. Add two **Marker2D** children to Enemy named "LeftBound" and "RightBound"
2. Position them where you want the enemy to turn
3. Use this code:

```
func _physics_process(delta):
    if not is_on_floor():
        velocity += get_gravity() * delta

    velocity.x = direction * SPEED

    # Check if reached left boundary
    if direction == -1 and global_position.x <= $LeftBound.global_position.x:
        direction = 1
        print("Reached left bound!")

    # Check if reached right boundary
    if direction == 1 and global_position.x >= $RightBound.global_position.x:
        direction = -1
        print("Reached right bound!")

    move_and_slide()
```

🎮 Controls Reference

Key	Action
←→	Move player
Spacebar	Jump
F5	Run game
F8	Stop game
Ctrl+S	Save current file
Ctrl+Shift+S	Save all files
Ctrl+D	Duplicate selected node

🔧 Troubleshooting

Problem	Cause	Solution
Enemy falls through platform	Wrong collision layers	Enemy mask must include Layer 4
Enemies stick together	Enemies colliding	Layer 2 should NOT be in enemy CharacterBody2D mask
Enemy doesn't move	SPEED = 0 or direction = 0	Check constants at top of script
Enemy doesn't turn	Timer not working	Check move_timer += delta line exists
Player doesn't die on contact	HitBox not configured	Check HitBox exists and signal connected
Level doesn't restart	Wrong function	Use call_deferred("reload_current_scene")
"Collision during physics" error	Not using call_deferred	Add call_deferred() before reload
Enemy doesn't detect player	Wrong collision mask	HitBox mask must include Layer 1
Enemy hovers/floats	No gravity	Check gravity code in _physics_process

---

📊 Success Checklist

Before finishing, verify:

- [ ] enemy.tscn scene created and saved
- [ ] Enemy has CollisionShape2D and ColorRect
- [ ] HitBox (Area2D) exists with CollisionShape2D
- [ ] Collision layers configured correctly:

    - [ ] Enemy CharacterBody2D: Layer 2, Mask 4
    - [ ] HitBox Area2D: Layer 2, Mask 1
    - [ ] Player: Layer 1, Mask 2 and 4
    - [ ] Platform: Layer 4
- [ ] Enemy patrols back and forth using timer
- [ ] Enemy turns every MOVE_TIME seconds
- [ ] Player dies when touching enemy
- [ ] Level restarts safely with call_deferred()
- [ ] Multiple enemies don't collide with each other
- [ ] No errors in Output panel
- [ ] All files saved (Ctrl+Shift+S)

🤔 Reflection Questions

Answer these in your exercise book:

1. **Why do we use collision layers and masks?**
   (Hint: What would happen if everything collided with everything?)
2. **How does the timer method work for patrol behavior?**
   (Hint: What happens when move_timer reaches MOVE_TIME?)
3. **Why is call_deferred() necessary when reloading the scene?**
   (Hint: What happens if we reload during physics calculations?)
4. **What does direction *= -1 do?**
   (Hint: What is 1 × -1? What is -1 × -1?)
5. **How would you make an enemy that patrols in a square pattern?**
   (Think: You'd need to change both X and Y movement...)
6. **What's the difference between the Enemy's collision and the HitBox's collision?**
   (Hint: One is for physics, one is for detection)

🎓 Key Concepts You Learned

Timer-Based AI

- Using delta time to count seconds
- Resetting timers for repeating patterns
- State management with direction variable

Collision System

- Layer separation for different object types
- Mask configuration for selective detection
- Physics callbacks and safe scene reloading

Code Patterns

- Multiplication for direction reversal (*= -1)
- Conditional logic for turning
- Signal connections for events
- Deferred function calls for physics safety

Game Design

- Challenge through enemies
- Consequences (death/restart)
- Fair vs unfair difficulty
- Predictable vs unpredictable patterns

📈 Difficulty Balancing Tips

Making Enemies Easier:

- Slower speed (SPEED = 50-75)
- Longer patrol time (MOVE_TIME = 3-4)
- Fewer enemies
- Wider platforms with safe spaces

Making Enemies Harder:

- Faster speed (SPEED = 150-200)
- Shorter patrol time (MOVE_TIME = 0.5-1)
- More enemies

- Narrower platforms
- Multiple enemies on same platform

Professional Game Design Principle:

"Difficult, but fair" - Players should always feel like they can succeed with skill and practice, not just luck!

Ask students: Is it fair if enemies are unavoidable? How much challenge is too much?

📝 Complete Enemy Script Reference

```
extends CharacterBody2D


# === MOVEMENT CONSTANTS ===
# How fast the enemy moves (pixels per second)
const SPEED = 100.0


# === MOVEMENT VARIABLES ===
# Current direction: 1 = moving right, -1 = moving left
var direction = 1


# Timer to track when to turn around
var move_timer = 0.0


# How long to move in each direction (in seconds)
const MOVE_TIME = 2.0


# === INITIALIZATION ===
# Called once when enemy is created
func _ready():
	print("=== ENEMY SPAWNED ===")
```

```
	# Connect HitBox signal if it exists
	if has_node("HitBox"):
		print("✓ HitBox found and connected!")
		# Connect the signal to detect player collision
		$HitBox.body_entered.connect(_on_hit_box_body_entered)
	else:
		print("✗ ERROR: HitBox not found!")

# === MAIN PHYSICS FUNCTION ===
# Called every frame (delta = time since last frame)
func _physics_process(delta):
	# --- APPLY GRAVITY ---
	# Make enemy fall if not on ground
	if not is_on_floor():
		velocity += get_gravity() * delta

	# --- MOVE IN CURRENT DIRECTION ---
	# Multiply direction (1 or -1) by SPEED to get velocity
	velocity.x = direction * SPEED

	# --- COUNT UP THE TIMER ---
	# Add elapsed time to the timer
	move_timer += delta

	# --- CHECK IF TIME TO TURN AROUND ---
	if move_timer >= MOVE_TIME:
		# Time's up! Reverse direction
```

```
            direction *= -1  # Flip: 1 becomes -1, -1 becomes 1
            move_timer = 0.0  # Reset timer back to zero
            print("Enemy turned! Now moving: ", "RIGHT" if direction == 1 else "LEFT")

    # --- EXECUTE THE MOVEMENT ---
    # Actually move the enemy based on velocity
    move_and_slide()

# === PLAYER COLLISION ===
# Called when something enters the HitBox
func _on_hit_box_body_entered(body):
    print("Something hit the enemy: ", body.name)

    # Check if "Player" is anywhere in the body's name
    # This works for: Player, Player2, PlayerL2, etc.
    if "Player" in body.name:
        print("💀 Player hit enemy! Restarting level...")
        # Use call_deferred to safely reload after physics finishes
        get_tree().call_deferred("reload_current_scene")
```

🎬 **Next Lesson Preview**

Lesson 5: Player Health & Lives System

Coming next:

- Health/hearts system (3 hits before death)
- Invincibility frames (i-frames) after taking damage
- Visual damage feedback (flashing effect)
- Respawn at starting position instead of full restart
- Game over screen

- Health UI display

Think about:

- How many hits should a player be able to take?
- What games have good health systems? (Mario, Zelda, etc.)
- Should enemies kill you instantly or just damage you?

🌟 Well Done!

You've created AI enemies with timer-based patrol behavior! Your game now has challenge and consequence.

What you achieved: ✅ Automated enemy movement
✅ Collision detection system
✅ Safe scene reloading
✅ Customizable difficulty

Next lesson: Make the game more forgiving with a health system instead of instant death!

📋 Peer Testing Form

Tester Name: _____
Game Creator: _____

Enemy Behavior

- [ ] Enemies patrol smoothly
- [ ] Enemies turn around regularly
- [ ] Multiple enemies work correctly
- [ ] Death on contact works

Difficulty Rating

☐ Too easy (enemies easily avoidable)
☐ Balanced (challenging but fair)
☐ Too hard (enemies nearly impossible to avoid)

Technical Quality

- [ ] No errors in Output panel

- [ ] Smooth movement (no jittering)
- [ ] Level restarts properly
- [ ] Enemies don't collide with each other

Bugs Found

1. _____
2. _____
3. _____

Suggestions for Improvement

_____

_____

Overall Rating

Gameplay: ☆ ☆ ☆ ☆ ☆
Challenge: ☆ ☆ ☆ ☆ ☆
Polish: ☆ ☆ ☆ ☆ ☆

End of Lesson 4 Quick Reference Guide

📚 **Teacher Notes**

Common Student Mistakes:

1. **Forgetting to add HitBox** - Code won't detect player
2. **Wrong collision layers** - Enemies collide with each other
3. **Not using call_deferred()** - Physics errors appear
4. **Player node named differently** - "Player" check fails

Extension for Advanced Students:

- State machine for multiple behaviors (patrol, chase, retreat)
- Path following with Path2D nodes
- Bezier curve patrol patterns
- Enemy health system (multiple hits to defeat)

Differentiation:

- **Support:** Pre-made enemy scene with just script needed
- **Challenge:** Implement chase behavior when player is nearby
- **Visual learners:** Draw patrol pattern on whiteboard first

Time Management:

- Scene creation: 8 minutes
- Collision setup: 7 minutes
- Script writing: 12 minutes
- Testing/debugging: 8 minutes
- Challenge activities: 5 minutes (if time permits)

Assessment Opportunities:

- **Formative:** Watch students configure collision layers
- **Summative:** Enemy behavior works correctly, code has comments
- **Peer assessment:** Use peer testing form