Lesson 2: Player Direction and Animation Setup

What We're Learning Today

- How to use if statements to make decisions in code
- How to store text in string variables
- How to track which direction the player is facing
- How to set up basic animations based on player direction

What We're Building

By the end of this lesson, your character will remember which direction they were last moving and display the correct idle animation when they stop. When you stop moving after going right, your character will face right. When you stop after going up, they'll face up!

Today's New Programming Concept

If Statements - If statements let your code make decisions. Think of them like asking questions: "IF the player pressed right, THEN make them face right." We use these to decide which animation to show.

Part 1: Understanding the Code

Step 1: Look at What We Have

Open your `scripts/player.gd` file and look at the `facing` variable:

```
var facing = "down"
```

This variable stores which direction our player is facing as text (called a "string"). Right now it always says "down", but we want to change it based on player movement.

Step 2: Guess What Will Happen

Before we code, think about this:

- When should we change the `facing` variable?
- What values should `facing` have? (Hint: "up", "down", "left", "right")
- How can we use the `xDirection` and `yDirection` values to know which way the player moved?

Step 3: Test Current State

Run your game from Lesson 1. Your character can move, but the `facing` variable never changes. Let's fix that!

Part 2: Learning If Statements

Simple Example

An if statement asks a question and does something based on the answer:

```
if xDirection > 0:
    print("Player is moving right!")
```

This reads as: "IF xDirection is greater than 0, THEN print that message."

How This Works

- `if` starts the question
- The condition (like `xDirection > 0`) is the question we're asking
- `:` ends the question
- Everything indented after the `:` happens when the answer is "yes"

Code Pattern for Multiple Choices

```
if xDirection > 0:
    # Player moving right
elif xDirection < 0:
    # Player moving left
elif yDirection > 0:
    # Player moving down
elif yDirection < 0:
    # Player moving up
```

`elif` means "else if" - it's another question to ask if the first one was "no".

Part 3: Building Direction Tracking

Step 1: Add Direction Detection

In your `_physics_process` function, after you get the input but before you set velocity, add code to update the facing direction.

You'll need to:

- Check if `xDirection` is greater than 0 (moving right)
- Check if `xDirection` is less than 0 (moving left)
- Check if `yDirection` is greater than 0 (moving down)
- Check if `yDirection` is less than 0 (moving up)
- Set the `facing` variable to the appropriate string: "right", "left", "down", or "up"

Hint: Use if/elif statements and remember that we only want to update facing when the player is actually moving (direction is not 0).

Step 2: Test Your Direction Tracking

Add a print statement to see the facing direction:

```
print("Player facing: ", facing)
```

Run your game and move around. You should see the facing direction change in the console:

- Move right, then stop → should print "Player facing: right"
- Move up, then stop → should print "Player facing: up"

Part 4: Setting Up Animations

Step 1: Understand Animation Names

Look at your player scene - the animations are named:

- "idle_down"
- "idle_up"
- "idle_left"
- "idle_right"

Notice the pattern: "idle_" + direction name.

Step 2: Create Animation Function

We'll create a helper function to set animations. Add this function to your script (outside of `_physics_process`):

```
func update_animation():
    # TODO: Set the animation based on the facing direction
    # Use: _animation_player.play("idle_" + facing)
    # This combines "idle_" with whatever direction we're facing
```

Step 3: Call Your Animation Function

In your `_physics_process` function, after you update the facing direction, call your new function:

```
update_animation()
```

Step 4: Test Animation Changes

Run your game and move around. Watch your character:

- Move right, then stop → character should face right
- Move up, then stop → character should face up
- The character should always show the correct idle pose

Part 5: Practice and Testing

Test 1: Direction Tracking

Move your character and watch the console output:

Expected behavior:

- Press RIGHT → facing changes to "right"
- Press UP → facing changes to "up"
- Press LEFT → facing changes to "left"
- Press DOWN → facing changes to "down"

Test 2: Animation Matching

Move your character and watch the sprite:

Expected behavior:

- When facing "right" → character sprite faces right
- When facing "up" → character sprite faces up
- Animation should match the console output

Test 3: Priority Testing

What happens when you press two directions at once (like UP and RIGHT)?

- Which direction wins?
- Is this the behavior you want?
- How might you change the if/elif order to get different priority?

Common Problems and Solutions

Problem: Facing direction doesn't change

Solution: Make sure your if statements are inside the `_physics_process` function and check that you're using `>` and `<` correctly.

Problem: Animation doesn't match direction

Solution: Check that your animation names match exactly: "idle_up", "idle_down", "idle_left", "idle_right".

Problem: Character faces wrong direction with diagonal movement

Solution: This is normal! Your if/elif order determines priority. The first condition that's true wins.

Problem: Facing changes even when not moving

Solution: You might be missing the condition to only update when actually moving (direction != 0).

What We Accomplished

🎉 Great work! You just learned:

- **If statements** to make decisions in your code
- **String variables** to store text like direction names
- **Logic conditions** using >, <, and == operators
- **Animation control** by combining strings to create animation names
- **Function creation** to organize your code better

Debugging Challenge

Add these print statements to better understand your code:

```
print("xDirection: ", xDirection, " yDirection: ", yDirection)
print("Player facing: ", facing)
print("Animation playing: idle_" + facing)
```

Run the game and see how the values change as you move around. This helps you understand the connection between input → direction → animation.

Next Time Preview

In our next lesson, we'll create our first collectible item - coins! We'll learn about collision detection and how to make objects disappear when the player touches them.