Lesson 1: Player Movement Basics

What We're Learning Today

- How to use variables to store numbers
- How to do basic math with multiplication
- How to get keyboard input from the player
- How to make our character move around the screen

What We're Building

By the end of this lesson, you'll be able to move your character up, down, left, and right using the arrow keys or WASD keys. When you press a key, your character will move smoothly across the screen!

Today's New Programming Concept

Variables - Variables are like labeled boxes that store information. In our game, we'll use variables to store numbers like how fast our player moves and which direction they're going.

Part 1: Understanding the Code

Step 1: Look at What We Have

Let's open the `scripts/player.gd` file and see what's already there:

```
extends CharacterBody2D
@onready var _animation_player: AnimatedSprite2D = $AnimatedSprite2D

var xSpeed = 300.0
var xDirection = 0
var facing = "down"
var ySpeed = 300.0
var yDirection = 0

func _physics_process(_delta):
    pass
```

Step 2: Guess What Will Happen

Before we start coding, let's think about what we see:

- What do you think `xSpeed = 300.0` means?
- What do you think `xDirection = 0` means?
- What do you think happens when `xDirection` is 0? What about when it's 1 or -1?
- Why do we have both `xSpeed` and `ySpeed`?

Step 3: Test and See

Right now, if you run the game (press F5), your character won't move at all because the `_physics_process` function is empty (it just has `pass` which means "do nothing").

Part 2: Learning Variables and Math

Simple Example

Think of variables like labeled jars:

- `xSpeed = 300.0` is like having a jar labeled "xSpeed" with the number 300 inside
- `xDirection = 0` is like having a jar labeled "xDirection" with the number 0 inside

How This Works

To make our player move, we need to do some simple math:

- **Speed** tells us HOW FAST to move (like 300 pixels per second)
- **Direction** tells us WHICH WAY to move (-1 for left, 0 for stopped, 1 for right)
- **Movement** = Speed × Direction (multiply speed by direction to get final movement)

Input.get_axis() Documentation

`Input.get_axis()` is a Godot function that checks two opposite keys and gives you a number:

How it works:

```
Input.get_axis("negative_key", "positive_key")
```

What it returns:

- `-1.0` when the negative key is pressed (like LEFT arrow)
- `1.0` when the positive key is pressed (like RIGHT arrow)
- `0.0` when neither key is pressed or both are pressed
- Values between -1 and 1 when using analog sticks (controllers)

Built-in key names:

- `"ui_left"` and `"ui_right"` - Left/Right arrow keys or A/D keys
- `"ui_up"` and `"ui_down"` - Up/Down arrow keys or W/S keys

Example:

```
var horizontal = Input.get_axis("ui_left", "ui_right")
# horizontal will be -1 (left), 0 (stopped), or 1 (right)
```

Code Pattern

```
```

```
# Get which direction the player wants to move
var direction = Input.get_axis("ui_left", "ui_right")

# Calculate how far to move by multiplying speed × direction
var movement = speed * direction

# Apply that movement to the player
velocity.x = movement
move_and_slide()
```

Part 3: Building Player Movement

Step 1: Get Input from the Keyboard

First, let's learn how to detect when the player presses keys. In your `_physics_process` function, replace `pass` with your code:

Step 2: Test Your Input Detection

Save your file and run the game (F5). Look at the bottom panel where it says "Output". Try pressing different keys:

- Press the RIGHT arrow key - you should see "X Direction: 1"
- Press the LEFT arrow key - you should see "X Direction: -1"
- Press the UP arrow key - you should see "Y Direction: -1"
- Press the DOWN arrow key - you should see "Y Direction: 1"
- Press nothing - you should see both directions as 0

Why does UP give us -1? In computer graphics, Y goes down as numbers get bigger, so -1 moves up the screen!

Step 3: Calculate Movement with Math

Now let's use multiplication to calculate how far to move. Add this code right after your print statements:

```
    #velocity is a vector which means it has a direction and a magnatude.
    #velocity.x is the x velocity
    #velocticy.y is the y velocity
```

Step 4: Apply the Movement

Finally, let's make the player actually move! Add this code at the end of your function:

```


    # TODO: Actually apply the movement
    # This is a special Godot function that makes the movement happen
```

```
    # Type this exactly: move_and_slide()


    # TODO: Print confirmation that we moved
    # Type this exactly: print("Player moved!")
```

Part 4: Practice and Testing

Test 1: Basic Movement

Save your file and run the game (F5). Try moving around with the arrow keys or WASD:

Expected Console Output when pressing RIGHT:

```
X Direction: 1
Y Direction: 0
X Movement: 300 Y Movement: 0
Player moved!
```

Expected Console Output when pressing UP and RIGHT together:

```
X Direction: 1
Y Direction: -1
X Movement: 300 Y Movement: -300
Player moved!
```

Test 2: Understanding the Math

Try this experiment:

1. Hold the RIGHT arrow key and watch the console
2. You should see "X Movement: 300" repeating
3. Now hold UP and RIGHT together
4. You should see "X Movement: 300 Y Movement: -300"

This proves our math is working: Direction × Speed = Movement!

Test 3: Change the Speed

Try changing the speed values at the top of your script:

```
var xSpeed = 100.0  # Change from 300 to 100
var ySpeed = 100.0  # Change from 300 to 100
```

Run the game again - notice how the player moves slower? Change it back to 300 when you're done testing.

Common Problems and Solutions

Problem: "Player moved!" prints even when not pressing keys

Solution: This is normal! The function runs every frame, even when the player isn't moving (direction = 0, so movement = 0).

Problem: Character moves too fast or too slow

Solution: Change the `xSpeed` and `ySpeed` numbers. Smaller numbers = slower, bigger numbers = faster.

Problem: UP and DOWN are backwards

Solution: This is normal in computer graphics! -1 moves up, +1 moves down.

Problem: Console is too cluttered with print statements

Solution: You can comment out some print statements by adding `#` at the beginning of the line.

What We Accomplished

🎉 Congratulations! You just:

- Used **variables** to store speed and direction numbers
- Used **multiplication** to calculate movement (Direction × Speed = Movement)
- Got **keyboard input** from the player
- Made your character **move smoothly** in all four directions
- Used **print statements** to see your math working

Your Complete Code

Next Time Preview

In our next lesson, we'll learn about **if statements** so we can track which direction our player is facing and set up animations for walking in different directions!