Lesson 4: Dangerous Spike Traps

What We're Learning Today

- How to apply collision detection to harmful objects
- How to create a health system for the player
- How to make objects affect player stats
- How to reuse code patterns from previous lessons

What We're Building

By the end of this lesson, your player will have health points and will take damage when touching spike traps. When you walk over spikes, you'll lose health and see your current health printed to the console. Be careful not to lose all your health!

Today's New Programming Concept

Player Stats and Variables - Games track information about the player like health, score, and lives using variables. We'll create a health system that can increase or decrease based on what happens in the game.

Part 1: Understanding the Code

Step 1: Look at What We Have

You already know about collision detection from the coin lesson! Spikes work the same way:

- They have an Area2D that detects when the player touches them
- They have a `_on_body_entered(body)` function that gets called automatically
- The difference is what happens when the player touches them

Step 2: Compare Coins vs Spikes

Think about the similarities and differences:

Coins:

- Player touches coin → coin disappears → player benefits

Spikes:

- Player touches spike → spike stays → player gets hurt

Both use the same collision detection, but the result is opposite!

Step 3: What We Need to Add

To make spikes work, we need:

1. A health system in the player
2. A way for spikes to damage the player

3. A way to track and display current health

Part 2: Learning Player Stats

Variables for Player Information

Just like we store `xSpeed` and `facing` in the player, we can store health:

```
var health = 100
var maxHealth = 100
```

Functions to Change Stats

We can create functions to modify player health:

```
func change_health(amount):
    # Change health by the amount (positive to heal, negative to damage)
    # Print the new health value
```

Why Use Functions?

Instead of changing health directly, functions let us:

- Add safety checks (health can't go below 0)
- Print debug information consistently
- Add effects later (like screen flash when hurt)

Part 3: Building the Health System

Step 1: Add Health Variables to Player

Open your `scripts/player.gd` file and add health variables near the other variables at the top.

You'll need:

- `var health` - current health points
- `var maxHealth` - maximum possible health

Hint: Start with values like 100 for both.

Step 2: Create a Health Change Function

Add a new function to your player script that handles changing health.

Your function should:

- Take a parameter for how much to change health (positive for healing, negative for damage)
- Add that amount to current health

- Make sure health stays between 0 and maxHealth
- Print the new health value

Function signature hint: `func change_health(amount):`

Step 3: Test the Health System

Add a temporary print statement in `_physics_process` to see your health:

```
print("Player Health: ", health)
```

Run the game - you should see your health value printing constantly.

Part 4: Making Spikes Dangerous

Step 1: Understand the Spike Script

The spike script (`spike.gd`) already has the collision detection set up:

```
extends Area2D

func _on_body_entered(body):
    pass
```

Notice it extends Area2D (just like coins) and has the same `_on_body_entered` function.

Step 2: Make Spikes Damage the Player

In the spike's `_on_body_entered` function, you need to:

- Check if the touching object is the player
- Call the player's health change function with negative damage
- Print a message about taking damage

Hints:

- Check `if body.name == "Player":`
- Call the health function on the body: `body.change_health(-10)` (negative for damage)
- Print something like "Player hit spikes!"

Step 3: Test Spike Damage

Run your game and walk over a spike. You should see:

- A message about hitting spikes
- Your health decrease by the damage amount
- Health continue to print in the console

Part 5: Understanding the Interaction

The Damage Flow

Here's what happens when you touch a spike:

1. Player enters spike's Area2D
2. Spike's `_on_body_entered` gets called automatically
3. Spike checks if it's the player
4. Spike calls `player.change_health(-10)`
5. Player's health decreases by 10
6. Player prints new health value

Why This Pattern Works

This follows good programming practice:

- **Spikes know how to detect collision** (their job)
- **Player knows how to manage health** (their job)
- **Each object handles its own responsibility**

Part 6: Practice and Testing

Test 1: Basic Damage

Walk over spikes and watch the console:

Expected behavior:

- Health decreases each time you touch a spike
- Damage message appears
- Health value updates immediately

Test 2: Multiple Spikes

Try touching different spikes:

- Each spike should work independently
- All spikes should deal the same damage
- Health should keep decreasing

Test 3: Health Limits

Keep taking damage until health reaches 0:

- Does your health stop at 0 or go negative?
- This tests if your damage function prevents negative health

Common Problems and Solutions

Problem: Health goes negative or above max

-what to do if health is above max health
-what to do if the health is <= zero.

Problem: Taking damage multiple times from one spike

Solution: This is normal! As long as you're touching the spike, you'll keep taking damage. We'll learn to fix this in later lessons.

Problem: "change_health" function not found error

Solution: Make sure you added the function to your player script, not the spike script.

Problem: Damage function never gets called

Solution: Check that the spike's signal is connected properly, just like with coins.

Advanced Challenge: Death System

If you want to experiment, try adding a death check:

```
func change_health(amount):
    # Your health change code here

    if health <= 0:
        print("Player died!")
        # Maybe reset health or restart level?
```

What We Accomplished

🎉 Outstanding work! You just learned:

- **Code reuse** - applying collision patterns from coins to spikes
- **Player stats** - tracking health with variables
- **Object interaction** - spikes affecting player state
- **Function parameters** - passing damage amounts between objects
- **Boundary checking** - preventing health from going negative
- **Responsibility separation** - each object handles its own job

Debugging Your Health System

Add these print statements to understand the flow:

```
# In spike script:
print("Spike touched by: ", body.name)

# In player health function:
print("Health changed by ", amount, ". New health: ", health)
```

This helps you see:

- When collision detection triggers
- What damage amounts are being applied
- How health changes over time

Next Time Preview

In our next lesson, we'll give our player the ability to fight back! We'll create projectiles that the player can shoot at enemies, learning about creating new objects dynamically and making them move across the screen.