Basic Game #2 GDD-Lite: The Ring Jump

Today, we'll be making a game where the "player" is an actual 2D character model (wowee!) that will move on the X axis across a plane with a cliff at the end. Pressing Spacebar makes your littleman jump, following the trajectory you were moving. Your littleman's goal? Jump through a ring down past the cliff!

Breaking it down

While making a coherent game (even a small one) seems daunting, they're usually just comprised of four parts: Objects; Mechanics; Physics; and Goals (yes, I will reuse this section in every GDD-Lite).

Objects:

Each individual physical entity on-screen at any point. These require references to one another so that when they interact, the game calls a mechanic. They also usually require collision shapes, which are how the game knows that objects are interacting with one another. But some elements, like UI elements, are designed not to have a collision shape so that objects just pass beneath them harmlessly.

- Character
- Ground (physical terrain)
- Ring
- UI Message

Mechanics:

Each individual interaction between objects whether it happens instantaneously; after a timer finishes; or via a physics calculation. These are usually the most memorable parts of a game.

- Pressing Left/Right (or A/D) moves the character left and right
- Pressing Spacebar makes the character jump
    - Jumping prevents the character from jumping while in the air
    - Once the character lands on the ground, they can jump again
- Leaving the screen at the bottom results in the MESSAGE being revealed
    - If the character successfully fell through the ring, you get a happy SOUND and the MESSAGE reveals that you won
    - If the character falls outside the ring, the lost MESSAGE and SOUND show up

Physics:

Each calculation that must be made to determine the speed, direction, and/or gravity of an object. These are typically calculated ahead of time and are called when a mechanic occurs, like jumping. However, some physics are also calculated only *after* or *during* a mechanic call (like a Pong ball getting faster over time or changing directions).

- Character moving on the X axis (left or right)
- Character jumping vertically
    - Character falling vertically
- Inability for Character to travel through the ground (physical terrain)

Goals:

The interaction that results in either a win condition or at least a step towards it. This is also the result of a mechanic occurring, but instead of causing something else to happen, it's usually the natural conclusion of a string of mechanical interactions when the player performs them correctly.

- Fall through the ring
    - Receive a good MESSAGE and SOUND

Now for the Coding

Now it's time to go through each object, mechanic, physics, and goal to detail the general code structure you might use in programming the game yourself! But this isn't your normal tutorial; we'll only be supplying code words or snippets that are relevant to get you started!

Objects:

- Character – First, you'll need to make a 2D BODY that can move on command.
    - Attach a SPRITE to it;
    - Create a HITBOX covering the SPRITE
- Ground (physical terrain) – You'll need to make a STATIC 2D BODY and:
    - Attach a SPRITE to it;
    - Create a HITBOX covering the SPRITE
    - The "Ground" will actually also be behind the player as a wall with the same properties
- Ring – You'll need to make a STATIC 2D BODY
    - Attach a SPRITE to it (this will actually be two attached sprites – more on that later);
    - Create a HITBOX that's ***smaller*** than the SPRITE
- Score UI (starting at 0) – You'll just need to make a MESSAGE for this and place it center-screen

Mechanics:

- Press Pressing Left/Right (or A/D) moves the character left and right
    - Call a SPRITE change during the PHYSICS PROCESS function
- Pressing spacebar jumps – You'll need four main functions here:
    - Bind a button or key to start the PHYSICS function detailed below;
    - Call a SPRITE change during the PHYSICS PROCESS function
    - Add an IF function to stipulate that if the Dart is moving, clicking does nothing, and if it's not moving, clicking calls the PHYSICS PROCESS function.
    - After the character hits the Ground, they can jump again
- Falling through the ring and off-screen rewards you with a winner UI and SOUND
    - We won't actually do anything here for now!

Physics:

- Character moving left and right
    - You'll need to add a PHYSICS PROCESS function that's called on your A/D presses
        - This function must STOP when the key is released
- Character moving vertically when initializing the jump
    - You'll need to add a PHYSICS PROCESS function to the Dart that moves it along the X axis at a constant speed;
        - This will be used to avoid moving through physical terrain

        - This will also be used to determine if the character passes through the ring
- Character stopping momentum when it hits physical terrain
    - Your character's PHYSICS PROCESS script will need a COLLISION variable to avoid passing through physical terrain and to determine if it passes through the ring
    - The PHYSICS PROCESS script must constantly check to see whether or not the character has COLLISION with another object, and then it calls the appropriate function

Goals:

- Fall through the ring
    - When falling through the ring, the character must pass on top of one SPRITE (the ring's back) and behind the other SPRITE (the ring's front) This can be achieved by setting the visual layer of all three sprites so they're different or just reordering each NODE!
    - You already have all the tools set up! You just need an IF function for whether the character has COLLISION with the ring before hitting the bottom of the stage!
        - That needs to call your win/loss MESSAGE and SOUND

Animation:

The first GDD-Lite didn't use any parts that required an animation. This time, we'll be using one to make the character stand still, walk, and jump. Now for this step, there are two main ways to do it: create an ANIMATION TREE (handy feature in Godot) which lets you easily blend two animation loops together; or simply calling the animation loop directly during certain mechanical or physics calls. We'll use the second one, since we don't need anything fancy.

- The first thing you need is your SPRITE. This should be created as a SPRITE sheet (an image file with each frame of a SPRITE's animation in order, placed equidistant apart from one another)
- You need to make the animation in an ANIMATION PLAYER within the engine; or
    - You need to create the animation loop itself manually (a hand-written script that cycles through each sprite sequentially) [not recommended]
- Then you just need to call that specific animation loop when the game state calls for it (a character moving; a character jumping; a fireworks effect; a ball compressing as it bounces)
    - You can call an h-flipped version of the animation, so you don't have to make it twice!
- The trickiest part is making sure that your 2D BODY script always calls back to the default stance/animation when other animations aren't happening (the idle animation!)
    - This can be tricky because the code is picky and unless it's written just right, a character's walk animation to the right may end and the character will automatically face left for the idle animation!
- The animation calls from your 2D BODY script

Resources:

We'd hate to make you find your own resources, so here's a handy zip file with the sprites and sounds you'll need!

Now you've made your game!