Assignment 5 – rendering and physics

This assignment will test your understanding of:

1. Custom canvas item shaders with particle systems
2. Advanced rigid body physics with joints
3. Raycasting with collision detection

Setup Instructions

1. Create a new Godot 4.x project with C# support
2. Create the following scene structure:
    - Main Scene (Node2D)
        - ParticleSystem (Node2D)
        - PhysicsDemo (Node2D)
        - LaserSystem (Node2D)
        - Player (CharacterBody2D)

Part 1: Custom Canvas Item Shader with Particles (3 points)

Requirements

Create a particle system with a custom shader that:

- Applies a color gradient effect
- Adds a wave distortion
- Responds to time for animation

Starter Code

ParticleController.cs

```
CSHARP
using Godot;

public partial class ParticleController : GpuParticles2D
{
    private ShaderMaterial _shaderMaterial;
```

```
    public override void _Ready()
    {
        // TODO: Load and apply custom shader
        // TODO: Configure particle properties (Amount, Lifetime, Speed, etc.)
        // TODO: Set process material properties

        // Hint: Use a new ShaderMaterial with your custom shader
    }

    public override void _Process(double delta)
    {
        // TODO: Update shader parameters over time
        // Hint: Use shader parameters to create animated effects
    }
}
```

custom_particle.gdshader (Create this file)

```
GLSL
shader_type canvas_item;

// TODO: Add uniform variables for:
// - Time-based animation
// - Color gradient (hint: use multiple color uniforms)
// - Wave intensity

uniform float wave_intensity = 0.1;
uniform vec4 color_start : source_color = vec4(1.0, 0.5, 0.0, 1.0);
uniform vec4 color_end : source_color = vec4(1.0, 0.0, 0.5, 1.0);

void fragment() {
    // TODO: Implement wave distortion using UV and TIME
    // TODO: Create color gradient based on UV.y
    // TODO: Apply final color with alpha

    // Starter hint:
    // vec2 uv = UV;
    // uv.x += sin(uv.y * 10.0 + TIME) * wave_intensity;
}
```

Part 2: Advanced Rigid Body Physics with Joints (3.5 points)

Requirements

Create a physics chain/rope system that:

- Uses multiple RigidBody2D nodes
- Connects them with PinJoint2D or DampedSpringJoint2D
- Responds realistically to player interaction
- Has at least 5 connected segments

Starter Code

PhysicsChain.cs

```
CSHARP
using Godot;
using System.Collections.Generic;

public partial class PhysicsChain : Node2D
{
    [Export] public int ChainSegments = 5;
    [Export] public float SegmentDistance = 30f;
    [Export] public PackedScene SegmentScene;

    private List<RigidBody2D> _segments = new List<RigidBody2D>();
    private List<Joint2D> _joints = new List<Joint2D>();

    public override void _Ready()
    {
        CreateChain();
    }

    private void CreateChain()
    {
        // TODO: Create chain segments
        // TODO: Position them appropriately
        // TODO: Connect them with joints
        // TODO: Configure joint properties (softness, bias, damping)

        // Hints:
        // - First segment should be StaticBody2D or pinned
        // - Use PinJoint2D.NodeA and NodeB to connect segments
        // - Set collision layers appropriately
    }

    // TODO: Add method to apply force to chain
    public void ApplyForceToSegment(int segmentIndex, Vector2 force)
    {
```

```
        // Apply impulse or force to specific segment
    }
}
```

ChainSegment.tscn (Create this scene)

- RigidBody2D
    - CollisionShape2D (RectangleShape2D)
    - Sprite2D or ColorRect for visualization

Part 3: Raycasting Laser with Player Detection (3.5 points)

Requirements

Create a laser security system that:

- Casts a ray continuously
- Visualizes the laser beam using Line2D
- Detects when the player crosses the beam
- Triggers an alarm (visual and/or audio feedback)
- Shows where the ray hits

Starter Code

LaserDetector.cs

```
CSHARP
using Godot;

public partial class LaserDetector : Node2D
{
    [Export] public float LaserLength = 500f;
    [Export] public Color LaserColorNormal = Colors.Green;
    [Export] public Color LaserColorAlert = Colors.Red;
    [Export] public NodePath PlayerPath;

    private RayCast2D _rayCast;
    private Line2D _laserBeam;
    private Node2D _player;
    private bool _isAlarmActive = false;
    private Timer _alarmTimer;

    public override void _Ready()
```

```
{
    SetupRaycast();
    SetupVisuals();
    // TODO: Get player reference
    // TODO: Setup alarm timer
}

private void SetupRaycast()
{
    // TODO: Create and configure RayCast2D
    // TODO: Set target position
    // TODO: Set collision mask to detect player
    // Hint: _rayCast = new RayCast2D();
}

private void SetupVisuals()
{
    // TODO: Create Line2D for laser visualization
    // TODO: Set width and color
    // TODO: Add points for the line
}

public override void _PhysicsProcess(double delta)
{
    // TODO: Force raycast update
    // TODO: Check if raycast is colliding
    // TODO: Get collision point
    // TODO: Update laser beam visualization
    // TODO: Check if hit object is player
    // TODO: Trigger alarm if player detected

    UpdateLaserBeam();
}

private void UpdateLaserBeam()
{
    // TODO: Update Line2D points based on raycast
    // Show full length if no collision
    // Show up to collision point if hitting something
}

private void TriggerAlarm()
{
    // TODO: Change laser color
```

```
        // TODO: Emit signal or call alarm function
        // TODO: Add visual feedback (flashing, particles, etc.)
        // TODO: Add audio feedback (optional)

        GD.Print("ALARM! Player detected!");
    }

    private void ResetAlarm()
    {
        // TODO: Reset laser to normal color
        // TODO: Reset alarm state
    }
}
```

Player.cs (Simple player for testing)

```
CSHARP
using Godot;

public partial class Player : CharacterBody2D
{
    [Export] public float Speed = 200f;

    public override void _PhysicsProcess(double delta)
    {
        // TODO: Implement basic movement (WASD or Arrow keys)
        // TODO: Use MoveAndSlide()

        Vector2 velocity = Velocity;

        // Get input and move

        Velocity = velocity;
        MoveAndSlide();
    }
}
```

Submission Requirements

What to Submit:

1. Githun Repository for the project
2. Screenshot/video showing:
    - Particle system with shader effects

    - Physics chain reacting to forces
    - Laser detecting player and triggering alarm
3. Repo should have a readme that explains:
    - How your shader works
    - Physics properties you chose and why
    - How the raycast detection works

Grading Rubric:

Part 1 - Shader & Particles (3points)

- Custom shader correctly applied
- Wave distortion working
- Color gradient implemented
- Time-based animation

Part 2 - Physics & Joints (3.5 points)

- Multiple rigid bodies created
- Joints properly configured
- Realistic physics behavior
- Player interaction working

Part 3 - Raycasting (3.5 points)

- Raycast correctly implemented
- Visual laser beam
- Player detection working
- Alarm system functional

Total: 10 points

Resources

- Godot Shading Language Documentation
- Godot Physics Documentation
- Godot Raycasting Tutorial