Interactive Justice Scale Asset Overview

The JusticeScale Asset provides a realistic and interactive scale system for Unity, designed for applications in simulations, mini-games, or any interactive scene that requires precise weight measurement and balance mechanics.

Use Cases

- Simulations: Integrate with physics-based objects for lifelike weight and balance behavior.
- Puzzle Games: Use weight-based interactions as a game mechanic, ideal for problem-solving or balance puzzles.
- Educational Tools: Demonstrate principles of weight and balance in interactive learning environments.

Table of Contents

1.Features

- **Realistic Weight Calculation**: Detects objects on the scale and calculates their total weight accurately.
- **Balance Display:** Visualizes weight differences through balance beam rotation, making it suitable for physics-based projects.
- **Customizable Sensitivity:** Configure how responsive the scale is to weight differences with adjustable parameters.
- **Demo Scene:** Includes a ready-to-use demo scene that demonstrates all primary features and allows for interactive testing.

2.Components

The asset contains four prefabs designed to facilitate weight measurement and balance simulation. Each prefab is tailored for specific use cases and functionalities, providing flexibility in implementation.

2.1 Prefabs Overview

1. **OverlapScale**
   This prefab represents a complete scale of justice model, including both sides of the scale. It includes two instances of the SimpleOverlapScale prefab, which are responsible for measuring weight using overlap capsule detection.
2. **SimpleOverlapScale**
   This prefab serves as a standalone weight measurement tool. It is designed for developers who require a simple scale to measure and return weight values using overlap capsule detection. It focuses solely on weight detection without additional features.
3. **TriggerScale**
   Similar to OverlapScale, this prefab also represents a complete justice scale model with both sides of the balance. However, instead of utilizing two SimpleOverlapScale instances, it incorporates SimpleTriggerScale instances for weight measurement via triggers. Like its overlap counterpart, it calculates the weight difference and displays the unbalanced side.
4. **SimpleTriggerScale**
   This prefab functions like SimpleOverlapScale, providing a straightforward method for measuring weight, but it uses triggers instead of overlap capsules. It is ideal for scenarios where trigger-based detection is preferred.

2.2 Advantages and Disadvantages of TriggerScale and OverlapScale

TriggerScale:

Advantages: Provides a high level of accuracy when detecting weight, as it consistently registers objects within its defined trigger area. This makes it particularly useful for scenarios where precise weight measurements are needed.

Disadvantages: The trigger area has a limited range, meaning objects outside of it won't be detected. Expanding the detection area by adding primitive colliders around the trigger zone is possible, but this can reduce precision. Maintaining accuracy over an expanded area may require adding specialized colliders to preserve measurement quality.

To increase its detection range, **TriggerScale** uses a capsule collider, which is less precise and harder to configure than other collider types, making it more difficult to fine-tune for specific use cases.

OverlapScale:

Advantages: Provides accurate detection of objects within a defined area using a capsule collider. The detection range can be adjusted by changing the length of the capsule, making it flexible for different object sizes.

Effective for reliably measuring objects and weight within the detection zone, offering good precision for larger objects.

Disadvantages: Using a capsule as the primitive collider shape can affect the detection accuracy, especially compared to a collider that more closely matches the exact shape of the object.

Although the capsule size can be adjusted, performance and measurement accuracy may decrease if a very large detection area is needed. In such cases, adding specialized colliders to maintain precision may be necessary.

3.Setup Instructions

3.1 Adding Scales to Your Scene

1. Import the Asset:
    - Import the Justice Scale asset into your Unity project by dragging and dropping the asset package into the Unity editor or using the package manager.
2. Drag Prefabs into the Scene:
    - Locate the prefabs in the Project window. You will find the following prefabs:
        - **OverlapScale:** This prefab contains a complete justice scale with two SimpleOverlapScale components to measure weight using overlap capsules.
        - **TriggerScale**: Similar to the OverlapScale, this prefab utilizes SimpleTriggerScale components to measure weight via triggers.
        - **SimpleOverlapScale:** A lightweight version designed to measure weight using overlap capsules without the complete scale setup.
        - **SimpleTriggerScale**: A simpler version for measuring weight using trigger colliders. It also has an additional capsule collider for greater range.
3. Positioning:
    - Drag the desired scale prefab (OverlapScale or TriggerScale) into your scene. Position it as needed to fit within your environment.

3.2 Configuring the Scales

1. Setting Up Scales:
    - Select the OverlapScale or TriggerScale prefab in the Hierarchy window. Ensure that both left and right scale references are assigned within the ScaleController component.
    - You can do this by dragging the appropriate SimpleOverlapScale or SimpleTriggerScale instances into the respective fields in the ScaleController inspector.
2. Display Configuration:
    - Each scale prefab comes with a UI display that shows the weight measurement. The Canvas containing this display is initially deactivated.
    - To enable the display, simply activate the Canvas object that is a child of the scale prefab. This will allow you to visualize the weight currently measured on the scales.

4.Code Overview

This section provides a detailed overview of the key scripts used in the Justice Scale asset, explaining their purpose, structure, and how they interact with each other.

4.1 ScaleController

The ScaleController script is responsible for managing the references to both scales in the balance and determining which side is heavier.

Key Features:

- Scale References: Contains public references to the left and right scales, ensuring that the balance operates correctly.
- Weight Calculation: Computes the weight difference between the two scales and updates the balance state accordingly.
- Balance Normalization: Normalizes the weight difference to provide a smooth transition for the balance animation.

4.2 ScaleBeamRotation

The ScaleBeamRotation script is responsible for rotating the balance beam based on the weight difference between the two scales.

Key Features:

- Beam Rotation: Calculates the rotation angle based on the BalanceNormalized value from the ScaleController.
- Visual Feedback: Provides visual feedback on the balance state, indicating which side is heavier.

4.3 OverlapScale and TriggerScale

The OverlapScale and TriggerScale scripts, which inherit from the parent class Scale, are responsible for measuring weight using overlap capsule detection and trigger colliders, respectively.

Key Features:

- Weight Measurement: Each scale computes the total weight based on the objects present on its side.
- Integration with ScaleController: Provides weight data back to the ScaleController to facilitate balance calculations.

4.4 SimpleOverlapScale and SimpleTriggerScale

These scripts serve as simplified versions of their respective scales, providing basic functionality for weight measurement without additional features.

Key Features:

- Lightweight Implementation: Focus on core weight measurement functionality, making them easy to integrate into custom setups.

5.Demo Explanation

The demo included with the Justice Scale asset provides an interactive experience showcasing the functionalities of both OverlapScale and TriggerScale. This demonstration allows users to experiment with the scales using various weights, helping to visualize the balance mechanics in action.

Features of the Demo:

- Instantiation: Click on the provided weight models to instantiate them onto the balance. Each weight has a different mass, allowing for diverse testing scenarios.
- Drag and Drop: Utilize the drag-and-drop functionality to place weights on either side of the scales. This interaction simulates real-world usage and provides immediate feedback on balance.
- Visual Feedback: Observe how the balance beam rotates to indicate which side is heavier. This visual representation enhances understanding of the balance dynamics at play.

To run the demo, simply open the provided scene in Unity. Additionally, users can test the scene in this web page or directly through the Unity editor. The demo serves as a practical example of how to utilize the scales effectively and can be modified to explore different functionalities and scenarios.

Link: https://play.unity.com/en/games/d027f202-e3ac-4ec6-ad40-563a829c179e/webgl-builds

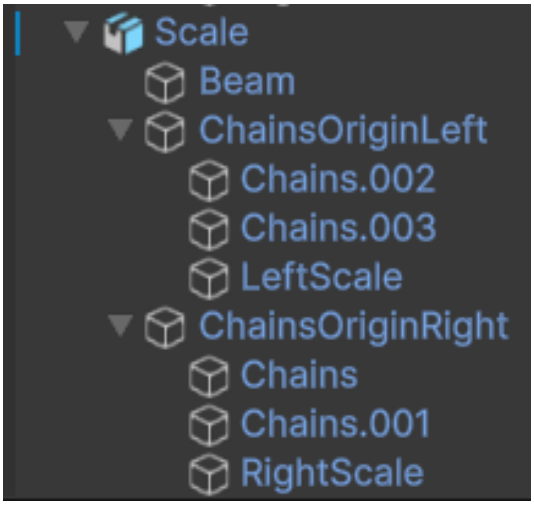
6.Using Your Own Balance Model

To integrate your own 3D model of a justice scale with the provided scripts, ensure the following requirements are met during the modeling process:

6.1 3D Model Hierarchy

The 3D model of a two-sided justice scale must consist of separate meshes arranged in a specific hierarchy to ensure compatibility with the scripts. The hierarchy should include the following components:

- **Base of the Balance:**
  This should be a standalone object that serves as the main support structure for the balance.
- **Beam:**
  The beam must be a separate object that can rotate. Make sure its pivot point is set correctly to allow for rotation around the appropriate axis.
- **Left Scale:**
  This must be an independent object, containing the hanging element (colgante) as a child. Ensure its pivot is aligned with the point of attachment to the beam.
- **Right Scale:**
  Similar to the left scale, this object should also be independent and have its hanging element as a child, with the pivot correctly positioned.
- **Plato (Plate):**
  This is the flat surface where objects are placed to be weighed. It can be part of the left and right scale models or designed as separate objects.

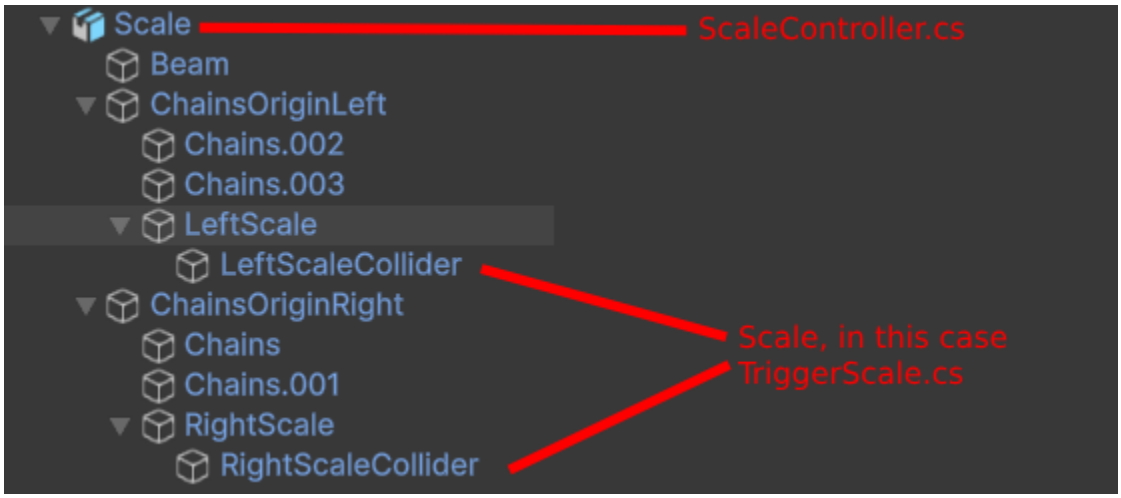


6.2 Modeling Considerations

- **Separate Meshes:** Each part of the balance (Base, Beam, Left Scale, Right Scale, their hanging elements, and the Plate) should be modeled as separate meshes to allow for proper interaction with the scripts.
- **Correct Pivot Points:** Ensure that the pivot points for rotation are accurately placed, particularly for the beam and the scales, to achieve the desired rotational behavior in Unity.
- **Logical Hierarchy:** While not directly applicable until imported into Unity, consider how these components will be connected and structured as GameObjects in Unity for optimal functionality.
- **Custom Collider for the Plate:** If desired, a special collider can be created for the plate of the balance (the surface where the items are placed for weighing) to enhance precision in detecting weights.
- **Plate:** make sure your plate where the objects will be weighed has its collisions.

6.3 Steps to Import and Configure in Unity

- **Import the Object:**
  Import your 3D model into Unity, ensuring that the hierarchy is preserved as described above. (You can use the importer preset)
- **Add Scripts to Scales:**
  Assign the appropriate script (either OverlapScale or TriggerScale) to each scale object (Left Scale and Right Scale).
- **Add ScaleController:**
  Attach the ScaleController script to the parent object (Base of the Balance) and assign the references to the left and right scales in the inspector.
- **Try using the prefabs as a reference:** As a tip, looking at the function in prefabricated can help you not waste a lot of time.

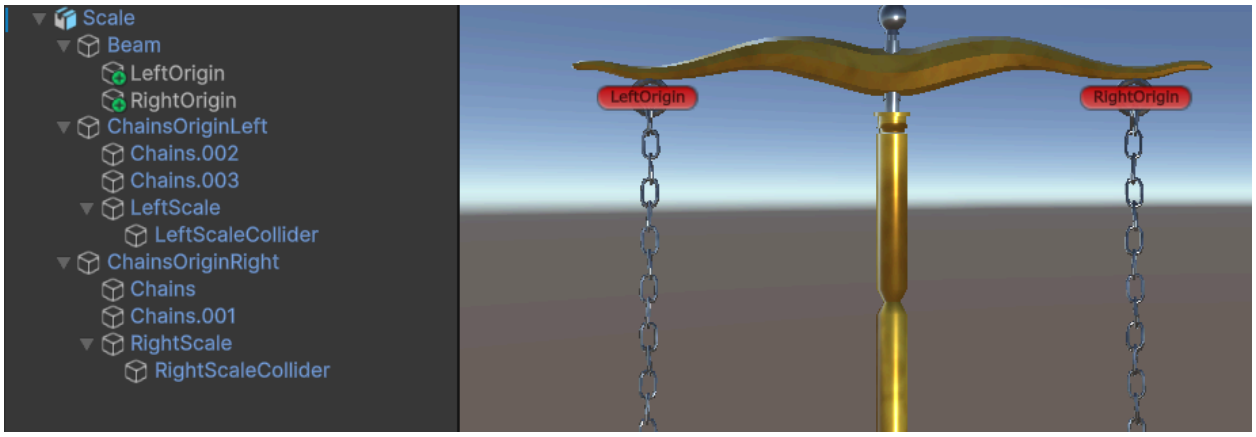


6.4 Configuring the Beam Rotation

To configure the ScaleBeamRotation correctly, follow these steps while maintaining the specified hierarchy:

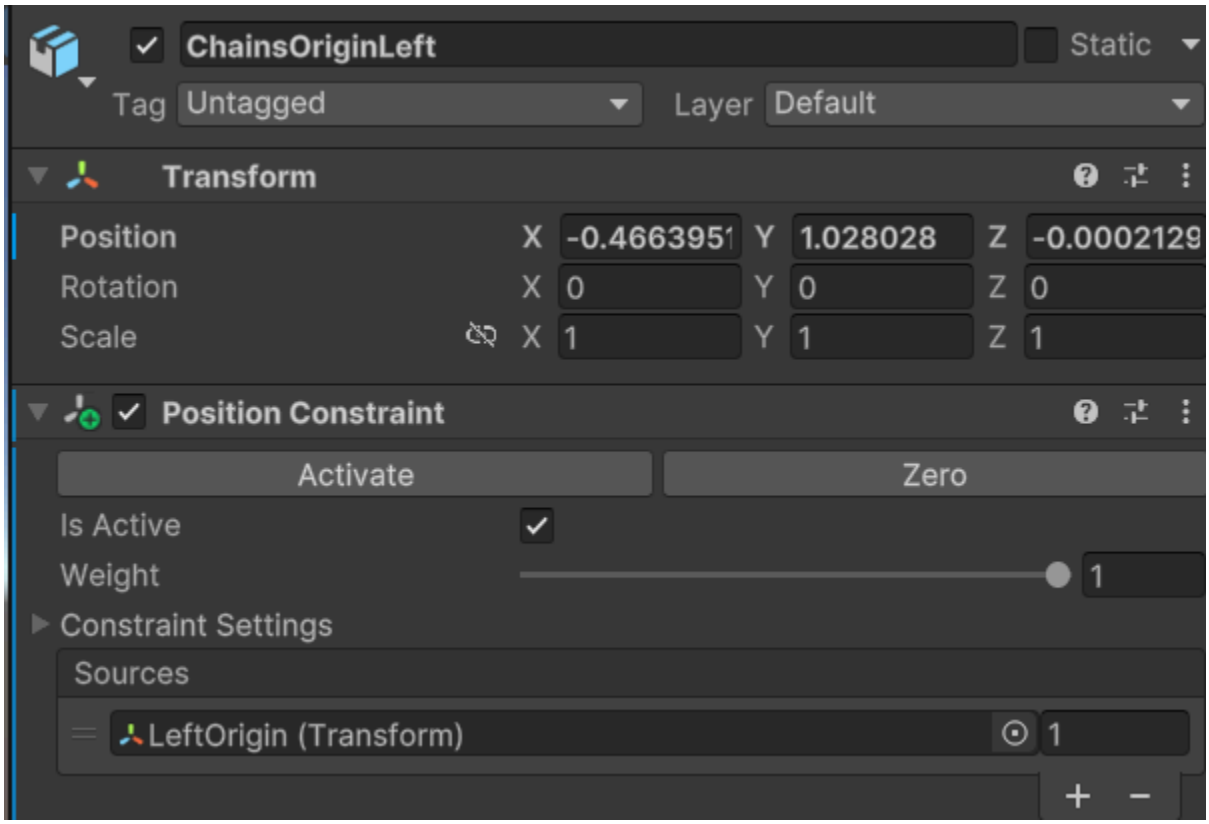
- **Beam**:
  The original beam object must have two empty child GameObjects positioned at the hanging origins of the left and right scales:
  - Left Hanging Origin: A child GameObject positioned at the origin of the left scale's hanging element.
  - Right Hanging Origin: A child GameObject positioned at the origin of the right scale's hanging element.

You have to put it at the origin of the scale bracket and then rename it as left origin or right origin, whatever makes sense to you. For the opposite side you can save steps by simply duplicating it, setting the opposite value to the x-axis position and finally renaming it.



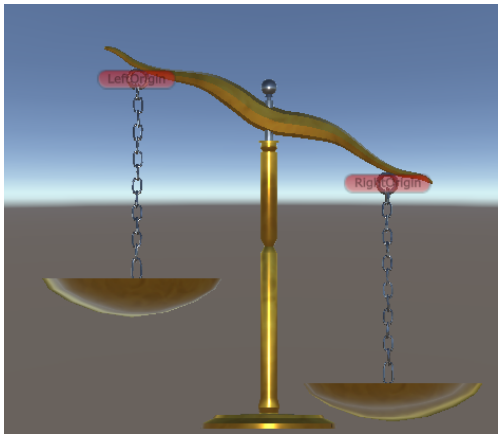
It is practically an obligation that they have the same height and depth so that the scale does not tilt.

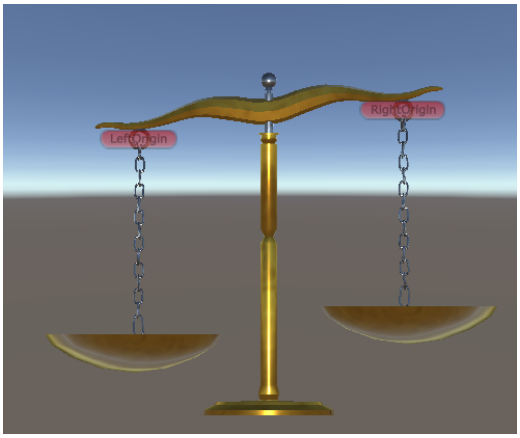
- **Left Scale and Right Scale:**
  Each scale must not be a direct child of the beam. If the scales are set as children of the beam, they will rotate along with it, leading to malfunctions in their behavior. Instead, the scales should be independent GameObjects that utilize the following method to maintain their position relative to the beam:
    - Attach a Position Constraint component to each scale. This component should be configured with references to the respective child objects of the beam. This setup ensures that the scales follow the height of the hanging origins without rotating with the beam.



Do the same with the other side. Sometimes the position constraint component can get buggy and a solution for me has been to press the zero and active buttons

Finally test it by rotating the beam, if both sides move with the rotation of the beam it is correctly configured. If you have them the other way around simply swap the references.





Also test what is the maximum rotation you would like without deforming to assign it to the blendRotation variable of the ScaleBeamRotation script.

- **Adding the Script:**
  Finally, add the ScaleBeamRotation script to the parent object where the ScaleControlle is and assign the beam transformation to it so that it works at runtime.