Golf Ball Controller 3d.

Table of contents

Overview:

- This asset allows the player to control the ball's direction and power by dragging and dropping the Mouse/Finger while displaying the UI elements to guide the player.
- Rotate the camera around the ball and follow it when the ball is moving using Mouse/Finger

Requirements:

- **Unity Version:** 2020.3.39f LTS.
- **Template type:** Built-in.

How to Install:

In the scene:

- Drag and drop the Floor Prefab into the scene.
- Drag and drop the Ball Prefab into the scene.
- The Ball Prefab should contain a child GameObject called "ui_direction".
- The ui_direction GameObject should have a Sprite Renderer component.
- Set the ui_direction's transform rotation along the x-axis to 90 degrees.

Set the components and its values of all GameObject in the scene as follows:

1. **Main camera :**
    a. Components:
        i. **Camera:**
            1. Set the field of view to get a better view.
        ii. **Script (CameraController.cs).**
            1. Assign the Ball's transform to the Target variable.
            2. Set the mouse sensitivity to a float number.
            3. Set the smooth time to a float number.
            4. Set the Rotation X min max to desired values.
2. **Directional light.**
    a. Set default values.
3. **Floor :**
    a. Components :
        i. **Box Collider.**
            1. Add the floor physics material from the Physics folder and set its values as desired.
        ii. **Material:** (floor).
4. **Ball :**
    a. Components :
        i. **Mesh Collider:**
            1. Use the ball mesh from Meshes folder.
        ii. **Sphere Collider:**
            1. Add Bounce Physics from Physics folder to it and set the values.
        iii. **Rigid Body**:
            1. Set the Mass to 1.
            2. Set the drag to 2.
            3. Set the Angular drag to 10.
            4. Set Collision Detection to Continuous.
        iv. **Line Renderer:**
            1. Material: (line).
            2. Adjust the width and color of the line as needed in the inspector.

        v. **Trail Renderer:**

1. Material: (trail).
2. Adjust the width and color of the trail as needed in the inspector.

vi. **Script (BallController.cs):**

1. Assign the **ui_direction** gameobject to the variable ui_direction in the inspector.
2. Set the maxDistance to 250.
3. Set Shoot Power to 0.2.

b. Ball Childs :

i. **ui_direction** :

1. Components:
   a. **Sprite Renderer:**
      i. Assign the ui_direction from the Texture folder.
   b. **Material:** (ui_direction).

Note:

- The **shader** for the materials (ball, floor, and trail) has been set to **Universal Render Pipeline/Lit** and **GPU Instancing** has been enabled.
- The **shader** for the materials (ui_direction and line) has been set to **UI/Default** and the **Render Queue** has been set to **Transparent**.
- In the Sprites folder, the **Texture Type** of (ui_direction, ui_selection, and line) has been set to
- **Sprite (2D and UI)** and the **Sprite Mode** has been set to **Single**.
- **Ui_direction** used to indicate the direction in which the ball will be shot.
- **Sprite size**: 512 x 512 png (Transparent).
- **Texture size**: 512 x 512 png (Transparent).
- Comments of each line are included in the script.

Scripts:

BallController.cs:

The script allows the player to control the ball's direction and power by dragging and dropping the mouse/Finger while displaying the UI elements to guide the player.

The purpose of the script is to allow the player to control the ball by using the functionality of drag and drop to shoot the ball in the preferred direction.

The script uses several components from the UnityEngine and UnityEngine.UI libraries, such as Rigidbody, GameObject, TrailRenderer and LineRenderer.

The script starts by defining the necessary objects using the [SerializeField] attribute to show them in the Unity Inspector. Then, it declares several variables to store the mouse position, ball status, distance, shoot, shootPower, angle, and cameraAngle, among others.

Start () method initializes the Rigidbody component of the ball and sets the camera angle to the Y-axis of the camera transform. It also gets the LineRenderer component, disables the UI direction gameobject.

OnMouseOver () method resets the distance to 0 and gets the start position of the mouse in world space. It also sets the rotation of the ball to zero and the camera angle to the Y-axis of the camera transform.

OnMouseDrag () method enables the UI. It also sets the isSelected flag to true and rotates the ball with the mouse rotation. Then, it converts the ball and mouse positions from world to screen point, calculates the distance between them, and sets up the shoot power. Finally, it gets the direction and angle between the ball position and the mouse position and changes the ball rotation accordingly, then Draw the line between the Ball and the mouse/finger.

OnMouseUp () disables ui_direction gameobject. Then, it sets isSelected variable to false, which could be used to indicate that the ball is no longer selected by the user. Next, it adds a force to the GameObject's rigid body component using the AddRelativeForce method, which applies a force in the forward direction of the GameObject's local space with a magnitude determined by the shoot scalar value. Finally, it calls a function named endLine to erase a line that was previously drawn between the ball position and the mouse/finger position.

drawLine (Vector3 startPos, Vector3 endPos) method sets up the LineRenderer to draw a line between two points (startPos and endPos) by setting the number of positions, creating an array of points, and setting the positions of the Line Renderer to the two points in the array.

endLine () method ends the line being drawn by setting the number of positions in the Line Renderer to 0.

CameraController.cs:

The script allows the player to rotate the camera around the ball using Mouse/Finger.

This code is for a camera controller in Unity that follows a target object and allows the user to rotate the camera around it using the mouse. Here is a breakdown of what the code does:

- **LateUpdate ()** method, which is called after all other Update () methods have been called in the current frame.

- The first if statement checks if the left mouse button is pressed or if the ball (presumably the target object is not selected.

- If the ball is selected, the camera's mouse sensitivity is set to 0 to allow for slower camera movement. Otherwise, the sensitivity is set to 2 for faster camera movement.

- The next two lines get the horizontal and vertical mouse movement and multiply it by the sensitivity value.

- The horizontal mouse movement is added to the current Y-axis rotation value, while the X-axis rotation value is set to 0 (presumably to prevent the camera from rotating up or down).

- The next line clamps the X-axis rotation value between the minimum and maximum values specified in the rotationXMinMax vector.

- A new rotation vector is created using the X-axis and Y-axis rotation values.

- The currentRotation vector is smoothly rotated towards the new rotation vector using Vector3.SmoothDamp() method, with the resulting rotation value stored in the currentRotation vector.

- The camera's position is set to the target object's position with an offset specified in the offset vector, while the camera looks at the target object using the LookAt () method.