Save this script, attach to the camera, assign the plane to the exposed variable *Layer* and try it out. It will work in a limited capacity in the Unity Editor with the mouse. The effect is far better experienced on a mobile device.

In a hidden cave, in the *Year Walk* forest, is a creepy wooden doll hanging from a rope. The player can tap on the doll to move it. Swiping its head causes the head to rotate around. After the correct number of rotations, more of the game clues are revealed to the player. This doll is a 3D model. To recreate this scene we will begin by creating a ragdoll in the next hands-on session.

Unity Hands-On

Creepy Head-Turning Doll

Step 1: Create a new Unity project and set for a mobile build. The camera can remain in the default perspective setting for this exercise. For a spooky look, add a single directional light and set the camera's background color to black. Download the *doll.fbx* and *hessian.png* from the website. Drag and drop these into the Asset window together. Unity will automatically assign the *hessian.png* texture to the doll model.

Drop the doll into the scene and position and orientate it such that it sits nicely in the view volume as shown in Figure 5.20. Ensure the doll's forward vector is facing toward the camera. With the doll selected in the Hierarchy, remove the Animator component in the Inspector.

Step 2: Select the doll in the hierarchy. Rename and reorganize all the parts of the doll such that they reflect the names shown in Figure 5.21. You will need to select each part to identify it on the model in the scene before renaming.

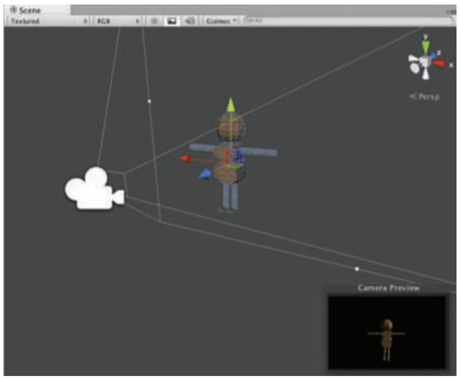


FIG 5.20 Adding doll model to Scene and positioning in the camera view.

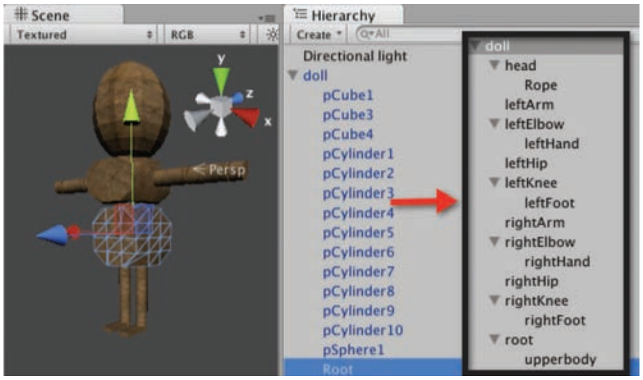


FIG 5.21 Renaming doll parts.

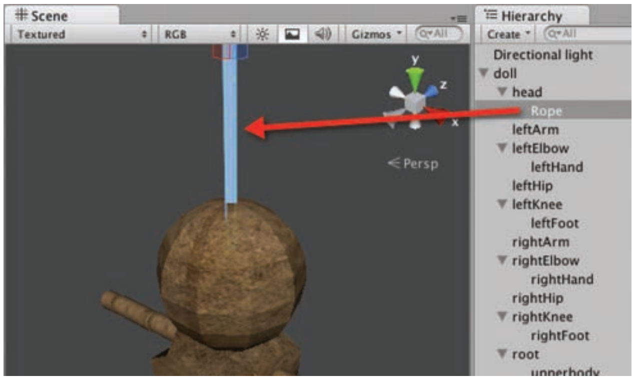


FIG 5.22 Adding a rope object to the doll game object.

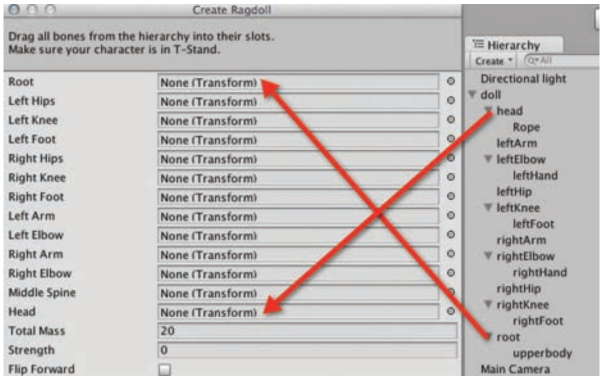
Note that the bottom half of the body is named the root.

Also note how some parts are parented to others. For example, the *leftFoot* is childed to the *leftKnee*. This is because we want the *leftFoot* to be physically attached to the *leftKnee* so that when the knee joint moves so does the foot. The *leftKnee* is not childed to the *leftHip* at this time as the *ragdoll* game object will take care of this later.

Create a cylinder with GameObject > Create Other > Cylinder and resize it to become a rope protruding from the top of the model's head as shown in Figure 5.22. Attach this cylinder to the head part of the doll model.

Step 3: GameObject > Create Other > Ragdoll. A dialog box will open requiring you to assign all the model parts for constructing a ragdoll as shown in Figure 5.23. Take each associated part that you have just renamed from the doll model and drag and drop into the associated positions.

When you are finished, select the Create button at the bottom of the popup and a ragdoll will be constructed from the model. Selecting the doll in the Hierarchy will reveal the joints and colliders added by the ragdoll constructor as shown in Figure 5.24.



Create Ragdoll

Drag all bones from the hierarchy into their slots.
Make sure your character is in T-Stand.

Root	None (Transform)
Left Hips	None (Transform)
Left Knee	None (Transform)
Left Foot	None (Transform)
Right Hips	None (Transform)
Right Knee	None (Transform)
Right Foot	None (Transform)
Left Arm	None (Transform)
Left Elbow	None (Transform)
Right Arm	None (Transform)
Right Elbow	None (Transform)
Middle Spine	None (Transform)
Head	None (Transform)
Total Mass	20
Strength	0
Flip Forward	☐

Hierarchy

Create

```
Directional light
▼ doll
  ▼ head
      Rope
    leftArm
  ▼ leftElbow
      leftHand
    leftHip
  ▼ leftKnee
      leftFoot
    rightArm
  ▼ rightElbow
      rightHand
    rightHip
  ▼ rightKnee
      rightFoot
    root
      upperbody
Main Camera
```

FIG 5.23 Ragdoll dialog box.

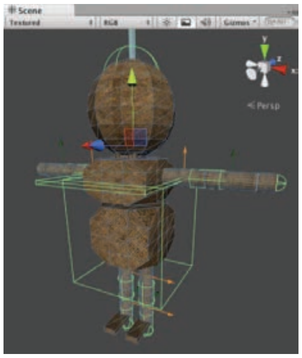


FIG 5.24 Ragdoll joints and colliders.

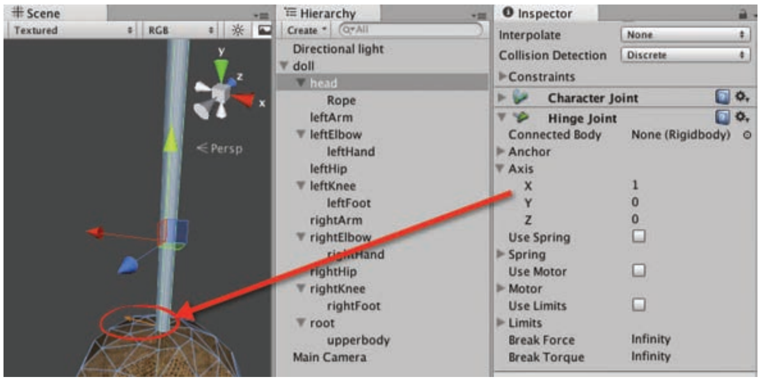
At this stage you can test the ragdoll by placing a plane beneath it to act as the ground and then pressing play. The ragdoll will fall under gravity and the physics system will control its movements.

Step 4: We will now suspend the ragdoll by the rope. Select the head in the Hierarchy and then Component > Physics > Hinge Joint.

Select the head in the Hierarchy and locate the attached Hinge Joint in the Inspector. The head will have a Character Joint (added by the Ragdoll creator) and a Hinge Joint. Make sure you locate the correct one.

The Hinge Joint will act as a suspender for the entire doll as the head is connected to the rest of the body by the ragdoll elements. The rope cylinder is just for show. You could remove it and the doll would react in the same way.

Note at the top of the head where the Hinge Joint has been added, there is a small orange arrow as shown in Figure 5.25. This arrow is defined in the Inspector of the Hinge Joint component and specifies the axis around



```
Directional light
▼ doll
  ▼ head
      Rope
    leftArm
  ▼ leftElbow
      leftHand
    leftHip
  ▼ leftKnee
      leftFoot
    rightArm
  ▼ rightElbow
      rightHand
    rightHip
  ▼ rightKnee
      rightFoot
  ▼ root
      upperbody
Main Camera
```

Interpolate	None
Collision Detection	Discrete
▶ Constraints	
▶ Character Joint	
▼ Hinge Joint	
Connected Body	None (Rigidbody)
▶ Anchor	
▼ Axis	
X	1
Y	0
Z	0
Use Spring	☐
▶ Spring	
Use Motor	☐
▶ Motor	
Use Limits	☐
▶ Limits	
Break Force	Infinity
Break Torque	Infinity

FIG 5.25 The small hinge joint arrow showing the axis of rotation.

which the joint will swing. In this case it is along the *x* axis. It will allow the doll to swing back and forth.

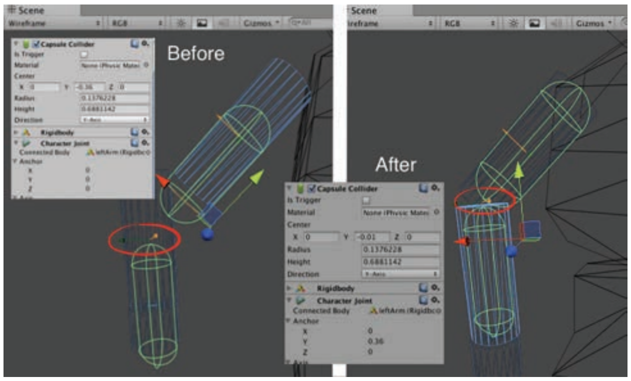
Press play and see how the doll reacts. The ragdoll components will be affected by gravity and collisions, but the whole model will still be suspended by the Hinge Joint.

You may notice that one of the ragdoll arms hangs a little funny. The reason being that the ragdoll creator doesn't always make a perfect ragdoll. However it has given you access to all the tools so you can make a few tweaks. The ragdoll works through a system of capsule colliders connected by anchor points. The anchor point is used to connect one collider to another. The issue with the doll's arm is that the hinge is located in the middle of the lower arm rather than up where the elbow should be as shown in Figure 5.26.

To fix this you can move the capsule collider further up into the arm using the Capsule Collider component's Y center value and relocate the joint's anchor up near the elbow. This is shown in the after image in Figure 5.26.

More Information: Physics Joints

For more information on the variety of physics joints in Unity see http://unity3d.com/learn/tutorials/modules/beginner/physics/joints.



Before:

```
Capsule Collider
Is Trigger
Material        None (Physic Mater
Center
X 0    Y -0.36    Z 0
Radius          0.1376228
Height          0.6881142
Direction       Y-Axis
Rigidbody
Character Joint
Connected Body  leftArm (Rigidbo
Anchor
X               0
Y               0
Z               0
```

After:

```
Capsule Collider
Is Trigger
Material        None (Physic Mater
Center
X 0    Y -0.01    Z 0
Radius          0.1376228
Height          0.6881142
Direction       Y-Axis
Rigidbody
Character Joint
Connected Body  leftArm (Rigidbo
Anchor
X               0
Y               0.36
Z               0
```

FIG 5.26 Adjusting ragdoll colliders and joints.

Step 5: To allow tap interaction with the doll, create a new C# script called *interact* and add the following code:

```
using UnityEngine;
using System.Collections;

public class interact : MonoBehaviour {

    void Update ()
    {
        if(Input.GetMouseButtonDown(0) || (Input.touchCount
        > 0 &&
            Input.GetTouch(0).phase == TouchPhase.Began))
        {
            RaycastHit hit;
            Ray ray = Camera.main.ScreenPointToRay(
                Input.mousePosition);

            if (!Physics.Raycast (ray, out hit, 10000))
                return;

            hit.rigidbody.AddForce(ray.direction * 1000);
        }
    }
}
```

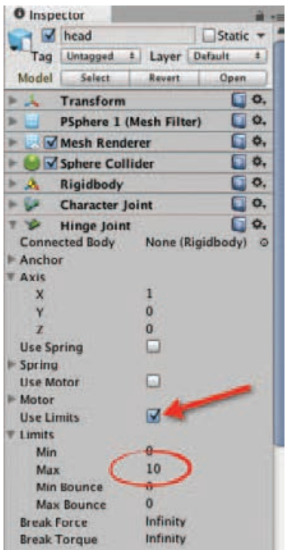
Save and attach to the main camera. Try it out. This code will allow you to click or tap on the doll and add a force to it in the same direction as the physics raycast from the touch-point. Because the doll contains numerous colliders, it is the collider that gets hit by the ray that has the force applied to it. The ragdoll setup of the model allows the physics from one part to affect the whole.

At this point you can get quite a swing up on the doll if you keep tapping on it. If you want to reduce the amount of movement you can limit the angle of the joints. Limits can be applied on all joints. To limit the Hinge Joint on the head to only allow it to swing backward to a maximum of 10 degrees apply the settings shown in Figure 5.27. Notice the Use Limits tick box needs to be ticked.

Try this out. You'll see a dramatic effect on the swing of the head. Remember the rest of the body doesn't have this restriction throughout so you'll still get a lot of movement.

Step 6: We are now going to add a head to the doll that can be spun around when a finger is dragged over it.

Create a basic sphere and add to the Scene. Locate it over the top of the existing head. The existing head is required by the ragdoll system and therefore instead of deleting it we are leaving it. You can, however, turn off or remove the existing head's Mesh Renderer so it doesn't interfere with the drawing of the new head. Name this sphere *SpinHead* and child it to the existing head as shown in Figure 5.28.



```
☑ head                ☐ Static
Tag Untagged          Layer Default
Model
PSphere 1 (Mesh Filter)
☑ Mesh Renderer
☑ Sphere Collider
Hinge Joint
Connected Body   None (Rigidbody)
Anchor
Axis
    X            1
    Y            0
    Z            0
Use Spring       ☐
Spring
Use Motor        ☐
Motor
Use Limits       ☑
Limits
    Min          0
    Max          10
    Min Bounce   0
    Max Bounce   0
Break Force      Infinity
Break Torque     Infinity
```

FIG 5.27 Limiting the swing of a Hinge Joint.

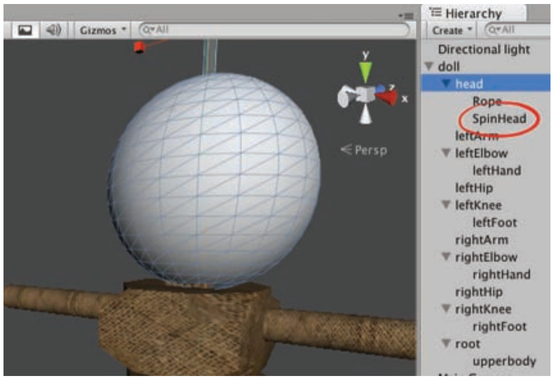
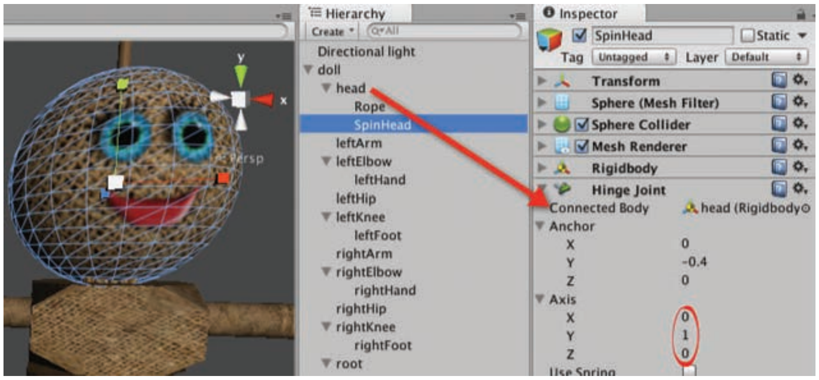


FIG 5.28 Adding an extra head layer to the ragdoll.



SpinHead Static
Tag Untagged Layer Default
Connected Body head (Rigidbody)
Anchor
X 0
Y -0.4
Z 0
Axis
X 0
Y 1
Z 0
Use Spring

FIG 5.29 Setup for the spinning head.

Step 7: To the *SpinHead* add a Rigidbody and a Hinge Joint. Set the angular drag of the *SpinHead* to 20. Locate the Hinge Joint component of the *SpinHead* and set the Connected Body property to be the original head. Also set the Hinge Joint axis to (0,1,0) as shown in Figure 5.29. There is also a texture file called *goodDolly.png* you can download from the website to put onto the head.

Setting the Hinge Joint axis to the *y* axis will allow it to spin around its up direction.

Step 8: To make the head spin around modify the `Update()` function of the *interact* script thus:

```
void Update ()
{
    if(Input.GetMouseButtonDown(0) ||
        (Input.touchCount > 0 &&
        Input.GetTouch(0).phase == TouchPhase.Began))
```

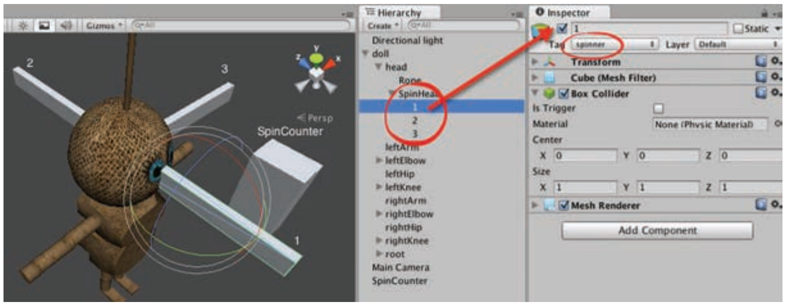
```
            {
                ...
            }
            else if((Input.touchCount > 0 &&
                Input.GetTouch(0).phase == TouchPhase.Moved))
            {
                RaycastHit hit;
                Ray ray =
                    Camera.main.ScreenPointToRay(Input.mousePosition);
                if (!Physics.Raycast (ray, out hit, 10000))
                    return;
                if(hit.rigidbody.gameObject.name == "SpinHead")
                {
                    hit.rigidbody.AddTorque(Vector3.up *
                    Input.GetTouch(0).deltaPosition.x * -10);
                }
            }
        }
```

This provides the ability to add a torque (circular) force to the *SpinHead* game object when a finger is dragged over the top of it. If the head spin is too fast or too slow for you, adjust the angular drag and/or the added torque.

Step 9: Last but not least, we are going to count the number of times the player spins the head and then, just because we can, after five spins, blow-up the doll. Begin by importing the detonator package used in Chapter 3. Now add four generic cubes to the scene. Reshape three of these cubes and position them coming out of the *SpinHead* as shown in Figure 5.30. Name them 1, 2, and 3 and make them children of the *SpinHead* object. Create a new tag called *spinner* and set it as the tag for these three blade objects.

Imagine each of these cubes as blades on a fan. As the *SpinHead* rotates, they rotate with it.

Now resize the fourth cube and position in just to the front right of the first blade. Make it big enough that it will be able to register each blade



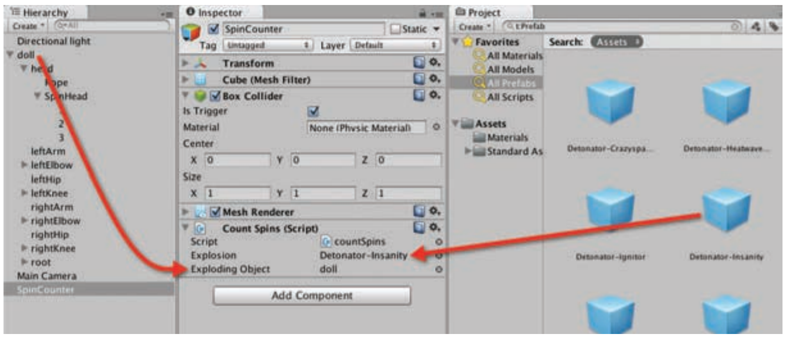
1 Static
Tag spinner Layer Default
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

FIG 5.30 Adding collider blades to a spinning object.

moving through it. Call it *SpinCounter*. Make its Box Collider a trigger by ticking its *isTrigger* tick box. Don't attach it to anything.

Step 10: As the head spins around—and thus the blades—we will be able to tell how the head is rotating by the sequence in which the blades hit the *SpinCounter*. For one full revolution to the right, the sequence would be blade 1 followed by 2, then 3, and then 1 again. Two revolutions to the right would give the sequence 1231231. We will now write some code to place onto the *SpinCounter* to keep track of the blade hit sequence. Create a new C# script called *countSpins* and add the following:

```
using UnityEngine;
using System.Collections;

public class countSpins : MonoBehaviour
{
    public GameObject explosion;
    public GameObject explodingObject;

    private string recordHits = "";
    private string matchString = "1231231231231231";

    void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.tag == "spinner")
        {
            recordHits + = other.gameObject.name;
            if(recordHits.Contains(matchString))
            {
                Instantiate(explosion,
                explodingObject.transform.position,
                Quaternion.identity);

                Destroy(explodingObject);
            }
            //clean out recordHits string to
            //stop it getting too big
            if(recordHits.Length >
            (matchString.Length * 2))
            {
                //remove first set of characters
                //that are the same length but
                //don't match the string
                recordHits = recordHits.Substring(
                    matchString.Length);

                   Debug.Log(recordHits);
            }
        }
    }
}
```



SpinCounter Static
Tag Untagged Layer Default
Transform
Cube (Mesh Filter)
Box Collider
Is Trigger ✓
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Mesh Renderer
Count Spins (Script)
Script countSpins
Explosion Detonator-Insanity
Exploding Object doll

FIG 5.31 Setup for an explosion to be triggered.

More Information: Strings

This code introduces the functionality of the C# .net string class with the Substring and Contains methods. For more detail on this see http://msdn.microsoft.com/en-us/library/system.string.aspx.

Save and attach the script to the *SpinCounter*. In the exposed public variables for Explosion and Exploding Object add the explosion prefab and the doll as shown in Figure 5.31.

Finally, you will want to turn off the Mesh Renderers for all four cubes so they are not visible in the final scene. Once you've done this, build to your mobile device and try it out.

The beauty of this script is that it will allow you to extend it to other puzzle-solving problems where the user must move or rotate objects in a certain order to achieve some goal. This could be attached to a locked safe in which the player has to rotate a dial back and forward in the right sequence to unlock. The `matchString` does all the work for you.

5.3 Drawing on the Screen

Drawing on the screen is an extension of finger-dragging, which uses the same sensing of finger-down, finger-moving, and finger-up but leaves behind actual evidence of the drag on the screen by way of a drawn line. The two games examined in this section implement this mechanic. First, *Fruit Ninja*