1 Super Rubber Ball

Step 1. Download and open the *SuperRubberBall* project from the website. Open the main scene. In it you will find a game track and a sphere as shown in Figure 1.1. The sphere has a *Rigidbody* added to it and the track has a mesh collider. Play. The sphere will fall onto the surface of the track.

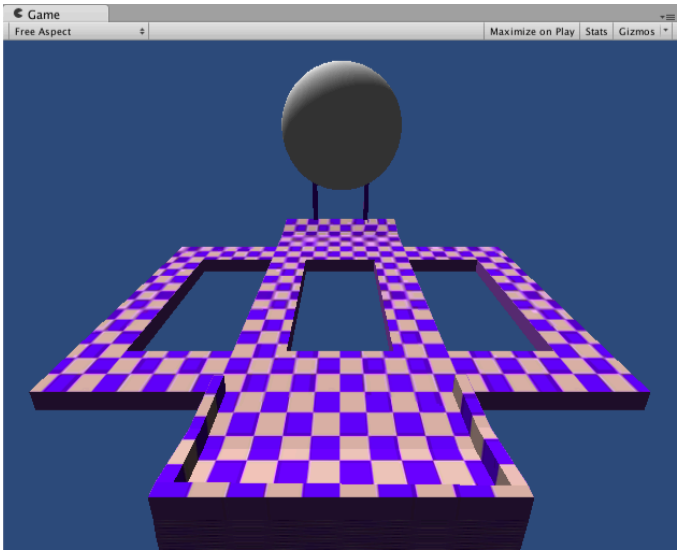

Figure 1.1 The starting scene for the SuperRubberBall game.

Step 2. Before we use the accelerometers of a device to tilt the track will use the arrow keys so we can debug the gameplay in the editor. Create a new C# script called *tiltTrack* and add the following code:

```
using UnityEngine;
using System.Collections;

public class tiltTrack : MonoBehaviour {

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update ()
	{

		if(Input.GetKey(KeyCode.RightArrow))
		{
			this.transform.Rotate(Vector3.forward * 1);

		}
		else if(Input.GetKey(KeyCode.LeftArrow))
		{
			this.transform.Rotate(Vector3.forward * -1);

		}
		else if(Input.GetKey(KeyCode.UpArrow))
		{
			this.transform.Rotate(Vector3.right * -1);
```

```
            }
            else if(Input.GetKey(KeyCode.DownArrow))
            {
                  this.transform.Rotate(Vector3.right * 1);

            }

      }
}
```

Save and attach the script to the *superballtrack* game object in the Hierarchy. Play. The arrow keys will tilt the track and the sphere will react appropriately.

Something you may notice is that if the sphere is allowed to fall and hit the track and then you start tilting it, the physics system will appear to not be working. It is not a glitch but rather that the physics system is ignoring your manual movements of the track model. To ensure the track is a physics object, select it in the Hierarchy and add a *Rigidbody* to it. In the track's new *Rigidbody* component ensure the `isKinematic` tickbox is checked. This will allow the track to cause physics events without being affected by them.

Note

If you find the sphere too hard to keep on the track, try changing the sphere's *Rigidbody* drag value to 5.

Step 3. To make the camera follow the sphere as it moves along the track we will create a 3D version of the smooth camera following algorithm used in Chapter 4. Create a new C# script called *cameraFollow* and add the following code:

```
using UnityEngine;
using System.Collections;

public class cameraFollow : MonoBehaviour {

      public GameObject target;
      Vector3 cameraTargetPos;
      Vector3 distanceFromTarget;

      void Start()
      {
            distanceFromTarget = new Vector3(
                  this.transform.position.x - target.transform.position.x,
                  this.transform.position.y - target.transform.position.y,
                  this.transform.position.z -
                        target.transform.position.z);
      }

      void Update ()
      {
            cameraTargetPos = new Vector3(
                  this.target.transform.position.x + distanceFromTarget.x,
                  this.target.transform.position.y + distanceFromTarget.y,
                  this.target.transform.position.z +
                        distanceFromTarget.z);
            transform.position = Vector3.Lerp(
                  this.transform.position,
                  cameraTargetPos, Time.deltaTime * 2.0f);
      }
}
```

Save and attach the script to the *Main Camera*. Set the `target` value of the script, in the Inspector, to the sphere.

Play, the camera will now smoothly follow the sphere keeping its relative distance. You may like to adjust the camera's position such that it is slightly above the sphere and looking down at it as shown in Figure 1.2.

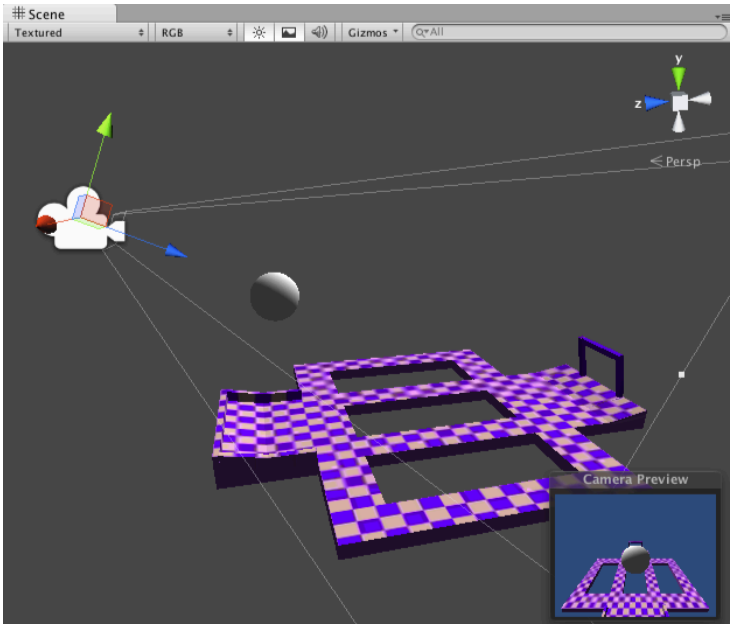

Figure 1.2 Positioning the camera relative to the sphere.

Step 4. Now that we have the basic movement mechanics in place its time to add the accelerometer. This requires two lines of code to access the accelerometer. Modify the *tileTrack* script thus:

```
...
public class tiltTrack : MonoBehaviour {

	public float sensitivity = 5.0f;
	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update ()
	{

		this.transform.Rotate(Vector3.forward *
			Input.acceleration.x/sensitivity);
		this.transform.Rotate(Vector3.right *
			Input.acceleration.z/sensitivity);

		if(Input.GetKey(KeyCode.RightArrow))
		{
			this.transform.Rotate(Vector3.forward * 1);
...
```

Save and built the game out to a mobile device for testing. Note, the code has a sensitivity adjustment for the accelerometer readings. The larger you make this value the slower the rotations will be applied.

Step 5. Next we will add some pickup items to the game. Create a cube and place it in the scene in front of the sphere. Add a material to it. In this example it will be shown as green. Give the cube a tag of "*GreenCube*", attach it to the *superballtrack* game object and set its Box Collider to be a trigger as shown in Figure 1.3.

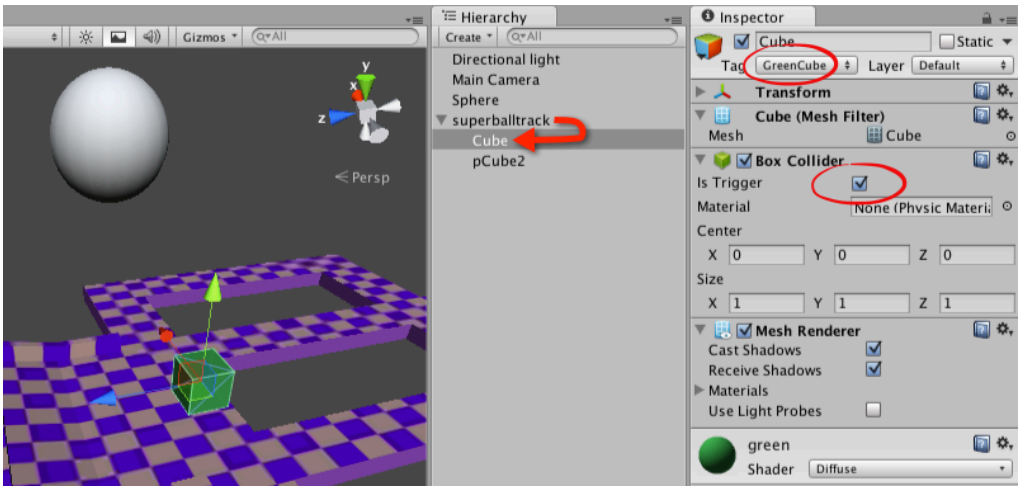

Figure 1.3 Adding a cube, that will become a pickup object, into the scene.

Step 6. Create a C# script called *gInterface* and add the following code:

```
using UnityEngine;
using System.Collections;

public class gInterface : MonoBehaviour
{
	static public int score = 0;

	void OnGUI()
	{
		GUI.Label(new Rect(10,10,100,20), "" + score);
	}
}
```

Save and attach to the Main Camera. When the game is run a *GUI Label* will display the score in the top left corner of the screen.

Step 7. Create a C# script called *pickup*. Add the following code:

```
using UnityEngine;
using System.Collections;

public class pickup : MonoBehaviour
{
	void OnTriggerEnter(Collider other)
	{
		if(other.gameObject.tag == "GreenCube")
		{
			gInterface.score++;
			Destroy(other.gameObject);
		}
	}
}
```

Save and attach to the sphere. Now when the sphere rolls over any objects that are triggers the pickup code will run and add one to the score before deleting the trigger object (in this case the green cube). Try it out.

A note about static variables

In the preceding code for `gInterface` the variable of score is declared as a static variable. The keyword *static* creates a variable that exists in the same location in computer memory for the entire run of a program. As such it allows us to access it using the name of the script that contains it and its name e.g. `gInterface.score`. You can only refer to static variables in this way. You should also ensure you have but one static variable of any name to avoid confusion.

Because static variables are in the computer's memory for the entire execution of your game you should ensure you only use them for items that need to exist for that long. Using them to store player statistics is perfect.

Step 8. Create a prefab from the *Cube* and remove it from the scene. Use the prefab to create multiple copies of the cube and scatter them throughout the scene. Remember to attach them to the track otherwise they will not move with the track as it tilts. Play. Roll the sphere over the cubes and watch your score increase.

Step 9. To end the level, the sphere must pass between the goalposts at the opposite end of the track. The pickup script that detects trigger objects can be added to detect the end of a level. First create a cube and place it just behind the goal posts as shown in Figure 1.4.

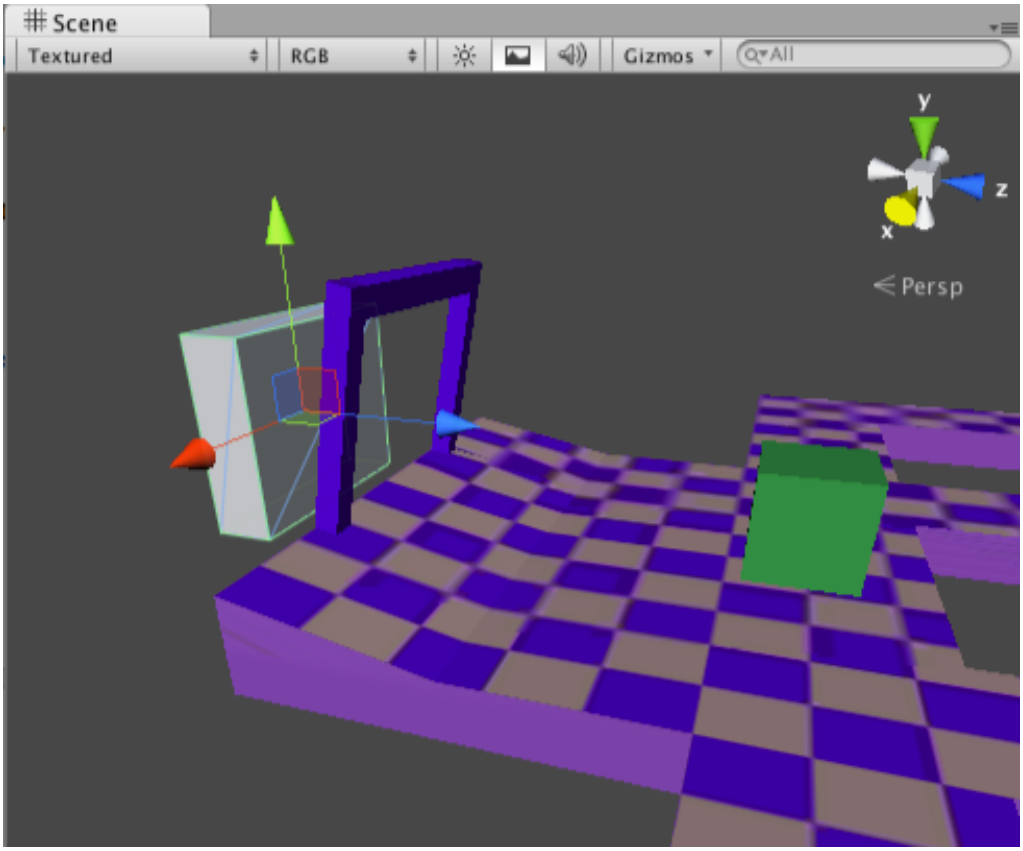

Figure 1.4 Adding a cube, that will trigger the end of the level.

Step 10. Make the cube a trigger and remove its mesh renderer. This will make it invisible. Attach it to the track. Give the cube a tag of "*LevelOver*".

Step 11. Modify the *gInterface* script like this:

```
...
static public bool levelOver = false;
```

```
void OnGUI()
{
      GUI.Label(new Rect(10,10,100,20), "" + score);
      if(levelOver)
      {
            GUI.BeginGroup (new Rect (Screen.width / 2 - 50,
                  Screen.height / 2 - 50, 100, 100));
            GUI.Box (new Rect (0,0,100,100), "Level Over");
            GUI.EndGroup ();
      }
}
...
```

Save. This new code will display a *GUI Box* in the middle of the screen when the level is over. The `levelOver` value is initially set to false in this script. In the next step, it is set to true by the sphere colliding with the end of level cube.

Step 12. Modify the pickup script thus:

```
public class pickup : MonoBehaviour
{
      void OnTriggerEnter(Collider other)
      {
            if(other.gameObject.tag == "GreenCube")
            {
                  gInterface.score++;
            }
            else if(other.gameObject.tag == "LevelOver")
            {
                  gInterface.levelOver = true;
            }
      }
}
```

Save and play. When the sphere passes through the goalposts at the end of the track, the level over GUI will display.