HOUR 10

Game 2: *Chaos Ball*

What You'll Learn in This Hour:

- How to design the game *Chaos Ball*
- How to build the *Chaos Ball* arena
- How to build the *Chaos Ball* entities
- How to build the *Chaos Ball* control objects
- How to further improve *Chaos Ball*

It is time once again to use what you have learned to make another game. In this hour, you'll make the game *Chaos Ball*, which is a fast-paced arcade-style game. You'll start by learning about the basic design elements of the game. From there, you'll build arena and game objects. Each object type will be made unique and given special collision properties. Then you'll add interactivity to make the game playable. You'll finish this hour by playing the game and making any necessary tweaks to improve the experience.

TIP

Completed Project

Be sure to follow along in this hour to build the complete game project. In case you get stuck, you can find a completed copy of the game in the book assets for Hour 10. Take a look at it if you need help or inspiration!

Design

In Hour 6, “Game 1: *Amazing Racer*,” you learned a lot about game design elements. This time, you’ll get right into them.

The Concept

This is a game slightly akin to *Pinball* or *Breakout*. The player will be in an arena. Each of the four corners will have a color, and four balls with corresponding colors will be floating around. Amid the four colored balls, there will be several yellow balls, called *chaos balls*. Chaos balls exist solely to get in the player’s way and make the game challenging. They are smaller than the four colored balls, but they also move faster. Players will have a flat surface with which they will attempt to knock the colored balls into the correct corners.

The Rules

The rules for this game state how to play and also allude to some of the properties of the objects. The rules for *Chaos Ball* are as follows:

- The player wins when all four balls are in the correct corners. There is no loss condition.
- Hitting the correct corner causes a ball to disappear and the corner light to go out.
- All objects in the game are super-bouncy and lose no energy on impact.
- No ball (or player) can leave the arena.

The Requirements

The requirements for this game are simple. This is not a graphically intense game; rather, it relies on scripting and interaction for its entertainment value. The requirements for *Chaos Ball* are as follows:

- A walled arena to play the game in.
- Textures for the arena and game objects. These are provided in the Unity standard assets.
- Several colored balls and chaos balls. These will be generated in Unity.
- A character controller. This is provided by the Unity standard assets.
- A game manager. This will be created in Unity.
- A bouncy physics material. This will be created in Unity.

- Colored corner indicators. These will be generated in Unity.
- Interactive scripts. These will be written in MonoDevelop.

The Arena

The first thing you want to create is an area where the action will take place. The term *arena* is used here to give the idea that this level is quite small and also walled in. Neither the player nor any balls should be able to leave the arena. Otherwise, the arena is quite simple, as shown in Figure 10.1.

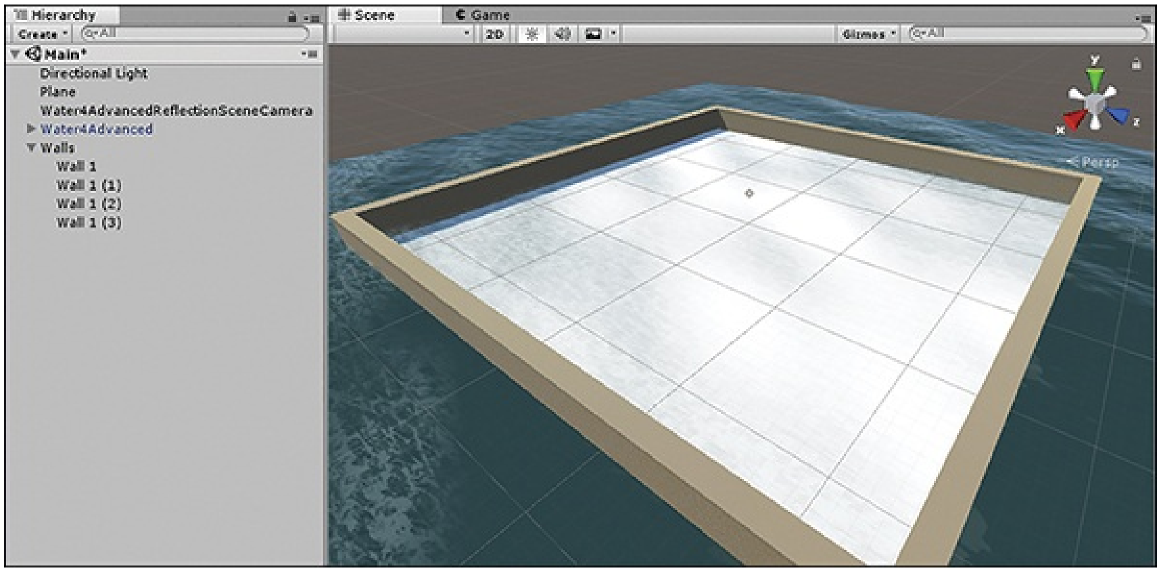


FIGURE 10.1
The arena.

Creating the Arena

As mentioned earlier, creating the arena is going to be easy because of the simplicity of the basic arena map. To create the arena, follow these steps:

1. Create a new project called Chaos Ball.
2. Click **Assets > Import Package** and select the **Characters** and **Environment** packages.
3. Add a plane to the scene (by selecting **GameObject > 3D > Plane**). Position the plane at (0, 0, 0) and give it a scale of (5, 1, 5).
4. Delete the Main Camera.

5. Add a cube to the scene. Place the cube at (–25, 1.5, 0) and scale it to (1.5, 3, 51). Notice how it becomes a side wall for the arena. Rename the cube **Wall 1**.

6. Save the scene as **Main** in a Scenes folder.

TIP

Consolidating Objects

You might be wondering why you created only a single wall when the arena will obviously need four. The idea is that you want to do as little redundant, tedious work as possible. Often, if you require several objects that are very similar, you can create one object and then duplicate it multiple times. In this instance, you set up a single wall with its materials and properties and then simply copy it three times. You repeat the same process for the corner nodes, the chaos balls, and the colored balls. Hopefully you can see that a little planning can save you a fair bit of time. It's also worth noting that this entire process can be made even easier with the use of prefabs. Since somebody (I won't mention names) doesn't cover prefabs until the next hour, you can just do it this way for now.

Texturing

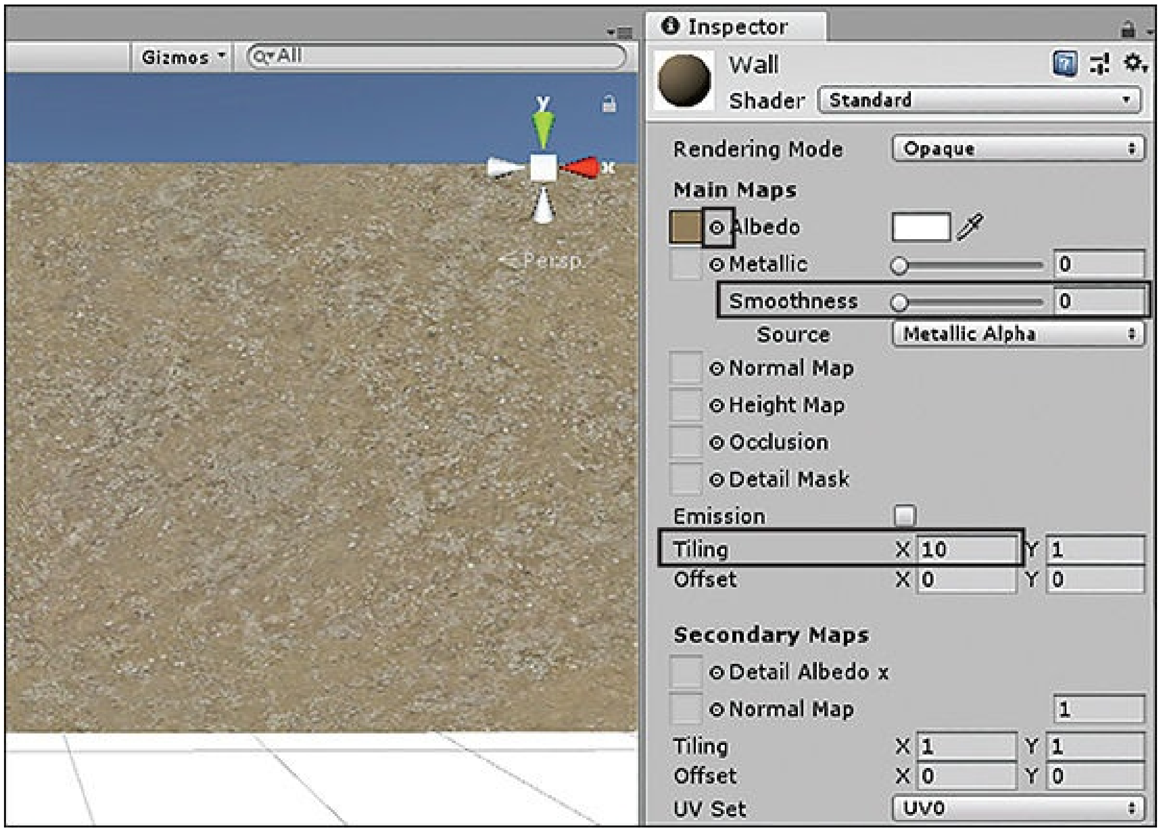
Right about now, the arena is looking pretty pitiful and bland. Everything is white, and there is only a single wall. The next step is to add some textures to liven the place up. You need to texture two objects specifically: the wall and the ground. Feel free to experiment with the texturing as you complete this step. You can make it more interesting if you'd like, but begin by following these steps:

1. Create a new folder called Materials under Assets in the Project view. Add a material to the folder (by right-clicking the folder and selecting **Create > Material**). Name the material **Wall**.

2. Apply the Sand Albedo texture to the Wall material in the Inspector view. You can do this by dragging the material onto the Albedo property or by clicking the circle selector next to the word Albedo in the Inspector (see Figure 10.2).

3. Drag the Smoothness slider to a value of **0**.

4. **4.** Set the X axis tiling to **10**.
5. **5.** Click and drag the wall material onto the wall object in the Scene view.



Wall
Shader Standard
Rendering Mode Opaque
Main Maps
Albedo
Metallic 0
Smoothness 0
Source Metallic Alpha
Normal Map
Height Map
Occlusion
Detail Mask
Emission
Tiling X 10 Y 1
Offset X 0 Y 0
Secondary Maps
Detail Albedo x
Normal Map 1
Tiling X 1 Y 1
Offset X 0 Y 0
UV Set UV0

FIGURE 10.2
The wall material.

Next, you need to make the ground more interesting. Unity comes with some great water shaders, and you can use them again here:

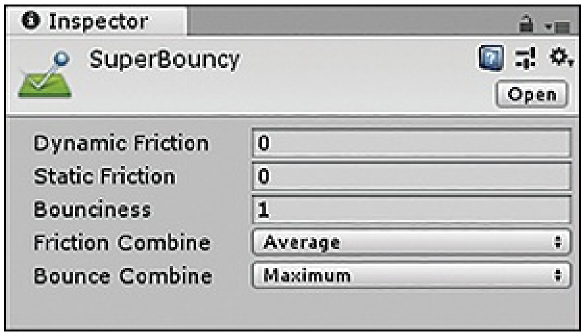
1. **1.** In the Project view, navigate to the folder Standard Assets\Environment\Water\Water4\Prefabs. Drag the Water4Advanced asset into the scene.
2. **2.** Position the water centrally, at (0, .5, 0).

Creating a SuperBouncy Material

You want objects to bounce off the walls without losing any energy, and what you need for this is a super-bouncy material. If you recall, Unity has a set of physics materials available. The bouncy material provided, however, is not quite bouncy enough for your needs. Therefore, you need to create a new material, as follows:

1. Right-click the **Materials** folder and select **Create > Physic Material**. Name the new material **SuperBouncy**.

2. Set the properties for the super-bouncy material as shown in Figure 10.3. Basically, you want to ensure that the balls are 100% bouncy, so they keep moving at the same speed.



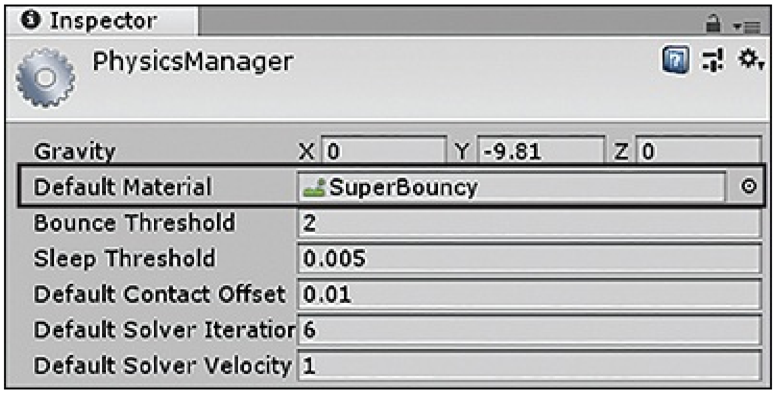
Dynamic Friction 0
Static Friction 0
Bounciness 1
Friction Combine Average
Bounce Combine Maximum

FIGURE 10.3
SuperBouncy material settings.

At this point, you could place the SuperBouncy physics material directly onto the collider of your wall. The problem, however, is that you will need to add this material onto all the walls, all the balls, and the player. Basically, everything that collides in this game needs this material. You can therefore apply the SuperBouncy material as the default material for all colliders by using the Physics Settings menu. Follow these steps:

1. Select **Edit > Project Settings > Physics**. The Physics Manager menu opens in the Inspector view (see Figure 10.4).

2. Apply the SuperBouncy material to the Default Material property.



Gravity X 0 Y -9.81 Z 0
Default Material SuperBouncy
Bounce Threshold 2
Sleep Threshold 0.005
Default Contact Offset 0.01
Default Solver Iteratior 6
Default Solver Velocity 1

FIGURE 10.4
The Physics Manager menu.

This menu is also where you can modify the fundamentals of how physics (collision, gravity, and so on) behave. For now though, you should resist the urge to be all-powerful and leave reality as it is.

Finishing the Arena

Now that the wall and ground are complete, you can finish the arena. The hard work has been done, and all you need to do at this point is duplicate the walls (by right-clicking in the Hierarchy view and selecting **Duplicate**). The exact steps are as follows:

1. Duplicate the wall once. Place the new instance at (25, 1.5, 0).
2. Duplicate the wall again. Place it at (0, 1.5, 25) with a rotation of (0, 90, 0).
3. Duplicate the wall created in step 2 (the one that's turned) and place it at (0, 1.5, –25).
4. Create an empty game object called **Walls**. Set the position of the new object to (0, 0, 0). Group the four walls you created under this new placeholder object.

Your arena should now have four walls without any gaps or seams (refer to Figure 10.1).

Game Entities

In this section you'll create the various game objects required for playing the game. Just as with the arena wall, it is easier to create one instance of each entity and then duplicate it.

The Player

The player in this game will be a modified First Person character controller. When you created this project, you should have imported the character controller's package. You will also want to raise the controller's camera up so that the player has a better field of vision while playing. Follow these steps:

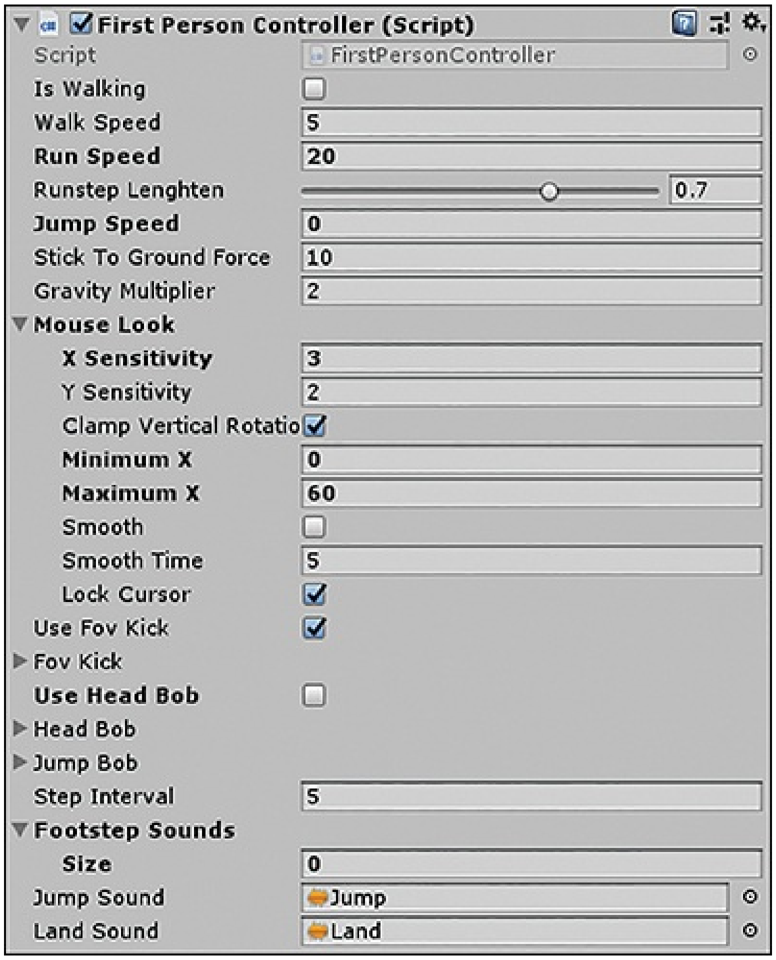
1. Drag an FPSController character controller into the scene, from the folder Assets\Characters\FirstPersonCharacter\Prefabs.
2. Position the controller at (0, 1, 0).

3. Expand the FPSController object in the Hierarchy view and locate the FirstPersonCharacter child object (which has a camera on it).
4. Position the FirstPersonCharacter at (0, 5, –3.5) with a rotation of (43, 0, 0). The camera should now be above, behind, and slightly looking down on the controller.

The next thing to do is to add a bumper to the controller. The bumper will be the flat surface off which the player will bounce balls. To do this, follow these steps:

1. Add a cube to the scene and rename the cube **Bumper**. Scale the bumper to (3.5, 3, 1).
2. In the Hierarchy view, click and drag the bumper onto the FPSController object to nest the bumper onto the controller.
3. Change the position of the bumper to (0, 0, 1) with a rotation of (0, 0, 0). The bumper is now slightly in front of the controller.
4. Give the bumper color by creating a new material (*not* a physics material) called BumperColor. Set the albedo color to something of your choosing and drag the material onto the bumper.

The last thing to do is to tweak the FPSController's default settings to make it more suitable for this game. Carefully set everything as per Figure 10.5, noting that the settings that differ from the defaults are bold.



First Person Controller (Script)

Script	FirstPersonController
Is Walking	☐
Walk Speed	5
Run Speed	20
Runstep Lenghten	0.7
Jump Speed	0
Stick To Ground Force	10
Gravity Multiplier	2
Mouse Look	
X Sensitivity	3
Y Sensitivity	2
Clamp Vertical Rotatio	✓
Minimum X	0
Maximum X	60
Smooth	☐
Smooth Time	5
Lock Cursor	✓
Use Fov Kick	✓
Fov Kick	
Use Head Bob	☐
Head Bob	
Jump Bob	
Step Interval	5
Footstep Sounds	
Size	0
Jump Sound	Jump
Land Sound	Land

FIGURE 10.5
The FPSController script settings.

The Chaos Balls

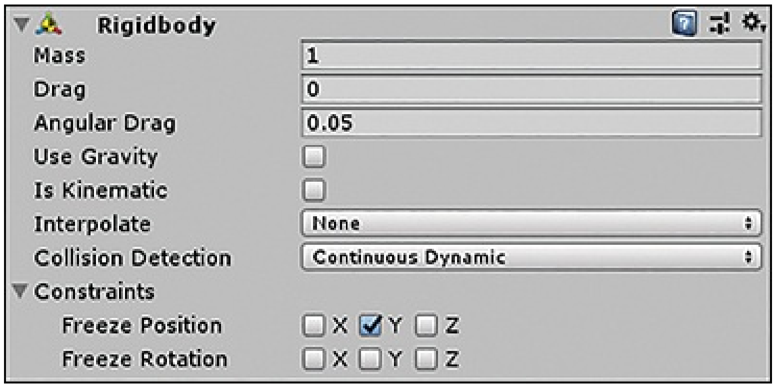
The chaos balls will be the fast and wild balls flying around the arena and disrupting the player. In many ways, they are similar to the colored balls, so you will be working to give them universally applicable assets. To create the first chaos ball, follow these steps:

1. Add a sphere to the scene. Rename the sphere **Chaos** and position it at (12, 2, 12) with a scale of (.5, .5, .5).

2. Create a new material (*not* a physics material) for the chaos ball called ChaosBall and set the albedo color to a bright yellow color. Click and drag the material onto the sphere.

3. Add a rigidbody to the sphere. As shown in Figure 10.6, uncheck **Use Gravity**. Change the Collision Detection property to **Continuous**

Dynamic. Under the Constraints property, freeze the y position because you don't want the balls to be able to go up or down.



Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	☐
Is Kinematic	☐
Interpolate	None
Collision Detection	Continuous Dynamic
Constraints	
Freeze Position	☐ X ☑ Y ☐ Z
Freeze Rotation	☐ X ☐ Y ☐ Z

FIGURE 10.6
The chaos ball rigidbody component.

4. Open the Tag Manager (by selecting **Edit > Project Settings > Tags & Layers**), expand the **Tags** section by clicking the arrow next to Tags, and add the tag **Chaos**. While you're here, go ahead and add the tags **Green**, **Orange**, **Red**, and **Blue** as well, as they are all used later.

5. Select the Chaos sphere and change its tag to **Chaos** in the Inspector view (see Figure 10.7).

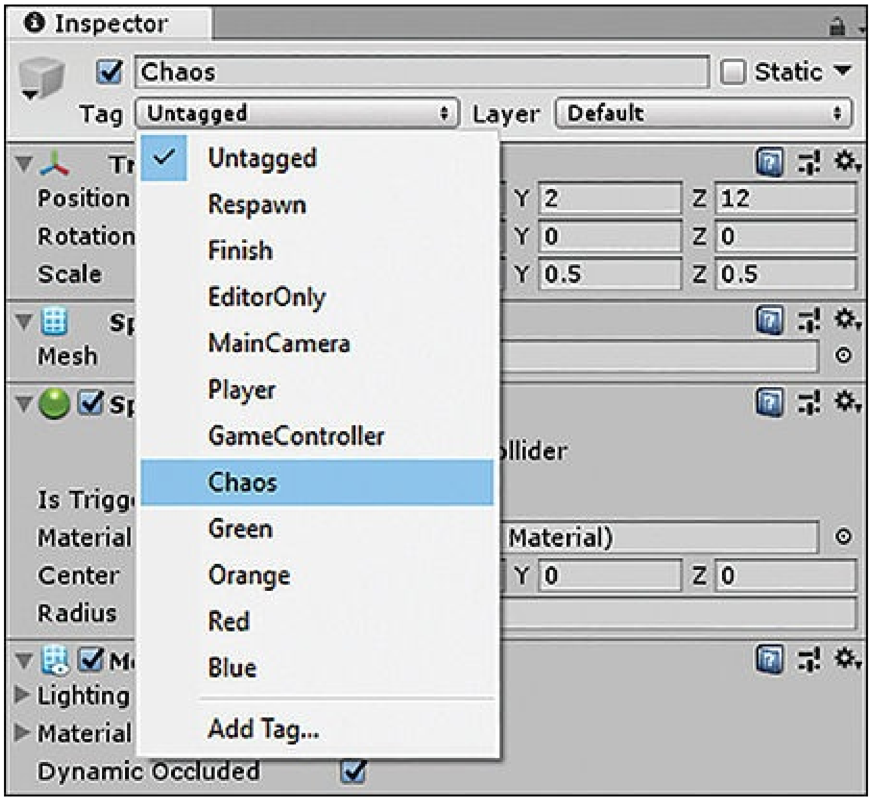


FIGURE 10.7
Choosing the Chaos tag.

The ball is now complete, but it still doesn't do anything. You need to create a script to move the ball all around the arena. In this case, create a script called VelocityScript and attach it to the chaos ball. Move the script into a Scripts folder. Listing 10.1 contains the full code for VelocityScript.

Listing 10.1 VelocityScript.cs

Click here to view code image

```
using UnityEngine;
public class VelocityScript : MonoBehaviour
{
    public float startSpeed = 50f;
    void Start ()
    {
        Rigidbody rigidBody = GetComponent<Rigidbody> ();
        rigidBody.velocity = new Vector3 (startSpeed, 0, startSpeed);
    }
}
```

Run your scene and watch the ball begin to fly around the arena. At this point, the chaos ball is finished. In the Hierarchy view, duplicate the chaos ball four times. Scatter each ball around the arena (be sure to only change the x and z positions) and give each of them a random y axis rotation. Remember that movement along the y axis is locked, so make sure that each ball stays at a y position of 2. Finally, create an empty GameObject called Chaos Balls, position it at (0, 0, 0), and child the balls to it to keep your Hierarchy tidy.

The Colored Balls

Whereas the chaos balls are actually a color (yellow), they are not considered colored balls in this game; rather, the colored balls are the four specific balls needed to win the game. They should be red, orange, blue, and green. As with the chaos balls, you can make a single ball and then duplicate it to make the creation of the colored balls easier.

To create the first ball, follow these steps:

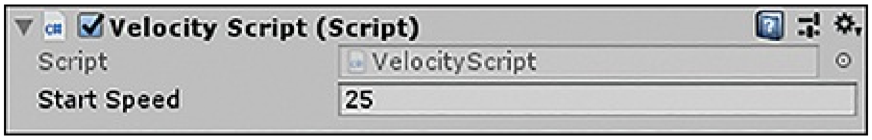
1. Add a sphere to the scene and rename it **Blue Ball**. Position the sphere somewhere near the middle of the arena and make sure the y position is **2**.

2. Create a new material called BlueBall and set its color to blue, the same way you set the color of the chaos balls to yellow. While you’re at it, go ahead and create RedBall, GreenBall, and OrangeBall materials and set each one to the appropriate color. Click and drag the BlueBall material onto the sphere.

3. Add a rigidbody to the sphere. Uncheck **Use Gravity**, set the collision detection to **Continuous Dynamic**, and freeze the y position under Constraints.

4. Previously, you created the Blue tag. Now change the sphere’s tag to Blue just as you set the tag for the chaos ball (refer to Figure 10.7).

5. Attach the velocity script to the sphere. In the Inspector, locate the Velocity Script (Script) component and change the Start Speed property to **25** (see Figure 10.8). This causes the sphere to initially move more slowly than the chaos balls.



Velocity Script (Script)

Script	VelocityScript
Start Speed	25

FIGURE 10.8
Changing the Start Speed property.

If you run the scene now, you should see the blue ball moving rapidly around the arena.

Now you need to create the other three balls. Each one will be a duplicate of the blue ball. To create the other balls, follow these steps:

1. Duplicate the Blue Ball object. Rename the new ball to its color: **Red Ball**, **Orange Ball**, or **Green Ball**.
2. Give the new ball a tag corresponding to its name. It is important for the name and the tag to be the same.
3. Drag the appropriate color material onto the new ball. It is important for the ball to be the same color as its name.
4. Give the ball a random location and rotation in the arena but ensure that its y position is **2**.

At this point, the game entities are complete. If you run the scene, you see all the balls bouncing around the arena.

The Control Objects

Now that you have all the pieces in place, it is time to gamify them. That is, it is time to turn them into a playable game. To do that, you need to create the four corner goals, the goal scripts, and the game controller. When you are done, you will have a game.

The Goals

Each of the four corners has a specific colored goal that corresponds with a colored ball. The idea is that when a ball enters a goal, the goal will check the ball’s tag. If the tag matches the color of the goal, there is a match. When a match is found, the ball is destroyed and the goal is set to solved. As with the ball objects earlier, you can configure a single goal and then duplicate it to match your needs.

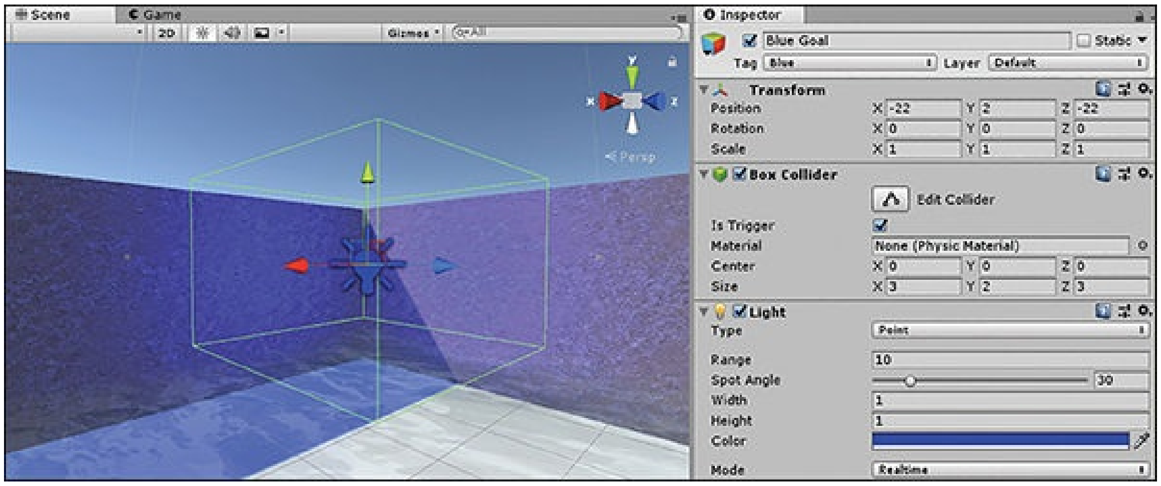
To set up the initial goal, follow these steps:

1. Create an empty game object (by selecting **GameObject > Create Empty**). Rename the game object **Blue Goal** and assign the tag Blue to

it. Position the game object at (–22, 2, –22).

2. Attach a box collider to the goal and check the **Is Trigger** property. Change the size of the box collider to (3, 2, 3).

3. Attach a light to the goal (by selecting **Component** > **Rendering** > **Light**). Make it a point light and make it the same color as the goal (see Figure 10.9). Change the Intensity of the light to **3** and the Indirect Multiplier to **0**.



Blue Goal ☐ Static
Tag Blue Layer Default

Transform

Position	X -22	Y 2	Z -22
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Box Collider

Is Trigger	✓		
Material	None (Physic Material)		
Center	X 0	Y 0	Z 0
Size	X 3	Y 2	Z 3

Light

Type	Point
Range	10
Spot Angle	30
Width	1
Height	1
Color	🟦
Mode	Realtime

FIGURE 10.9
The blue goal.

Next, you need to create a script called GoalScript and attach it to the blue goal. Listing 10.2 shows the contents of the script.

Listing 10.2 GoalScript.cs

Click here to view code image

```
using UnityEngine;
public class GoalScript : MonoBehaviour
{
    public bool isSolved = false;
    void OnTriggerEnter (Collider collider)
    {
        GameObject collidedWith = collider.gameObject;
        if (collidedWith.tag == gameObject.tag)
        {
            isSolved = true;
            GetComponent<Light>().enabled = false;
            Destroy (collidedWith);
        }
    }
}
```

As you can see in the script, the `OnTriggerEnter()` method checks the tag of every object that contacts it against its own tag. If they match, the object is destroyed, the goal is flagged as solved, and that goal's light is disabled.

When the script is complete and attached to the goal, it is time to duplicate it. To create the other goals, follow these steps:

1. Duplicate the Blue Goal. Name the new goal according to its color: **Red Goal**, **Green Goal**, or **Orange Goal**.
2. Change the tag of the goal to its corresponding color.
3. Change the color of the point light to the goal's corresponding color.
4. Position the goal. The colors can go in any corner as long as each goal gets its own corner. The three other corner positions are (22, 2, –22), (22, 2, 22), and (–22, 2, 22).
5. Organize the goals under a new empty game object called **Goals**.

All the goals should now be set up and operational.

The Game Manager

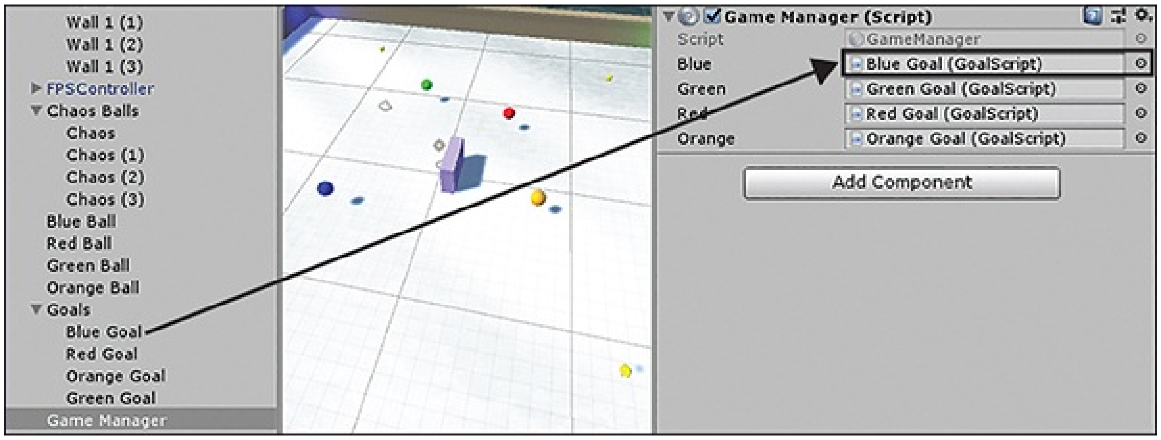
The last element needed to finish the game is the game manager. This controller will be responsible for checking each goal every frame and determining when all four goals are solved. For this particular game, the game manager is very simple. To create the game manager, follow these steps:

1. Add an empty game object to the scene. Move it someplace out of the way. Rename it **Game Manager**.
2. Create a script called GameManager and add the code from Listing 10.3 to it. Attach the script to the game manager.
3. With the game manager selected, click and drag each goal to its corresponding property on the Game Manager (Script) component (see Figure 10.10).

Listing 10.3 Game Control Script

Click here to view code image

```
using UnityEngine;
public class GameManager : MonoBehaviour
{
    public GoalScript blue, green, red, orange;
    private bool isGameOver = true;
    void Update ()
    {
        // If all four goals are solved then the game is over
        isGameOver = blue.isSolved && green.isSolved && red.isSolved &&
        orange.isSolved;
    }
void OnGUI() {
        if(isGameOver)
        {
            Rect rect = new Rect (Screen.width  2 - 100, Screen.height
            GUI.Box (rect, "Game Over");
            Rect rect2 = new Rect (Screen.width  2 - 30, Screen.height
            GUI.Label (rect2, "Good Job!");
        }
    }
}
```



Game Manager (Script)
Script GameManager
Blue Blue Goal (GoalScript)
Green Green Goal (GoalScript)
Red Red Goal (GoalScript)
Orange Orange Goal (GoalScript)

FIGURE 10.10
Adding the goals to the game controller.

As you can see in the script shown in Listing 10.3, the game manager has a reference to each of the four goals. Every frame, the manager checks to see if all the goals are solved. If they are, the manager sets the variable `isGameOver` to `true` and displays the “game over” message on the screen.

Congratulations. *Chaos Ball* is now complete!

Improving the Game

Even though *Chaos Ball* is a complete game, it is hardly as good as it could be. Several features that would greatly improve gameplay have been omitted. They were left out for brevity and to give you an opportunity to experiment with the game and make it better. In a way, you could say that *Chaos Ball* is now a complete prototype. It is a playable example of the game, but it lacks polish. You are encouraged to go back through this hour and look for ways to make the game better. Think about the following as you play it:

- Is the game too easy or hard?
- What would make it easier or harder?
- What would give it a "wow" factor?
- What parts of the game are fun? What parts of the game are tedious?

The exercise at the end of this hour gives you an opportunity to improve the game and add some features that will improve it. Note that if you get any errors, it means you missed a step. Be sure to go back through and double-check everything to resolve any errors that arise.

Summary

In this hour, you made the game *Chaos Ball*. You started by designing the game, based on the stated concept, rules, and requirements. From there, you sculpted the arena and learned that sometimes you can make a single object and duplicate it to save time. From there, you created the player, the chaos balls, the colored balls, the goals, and the game controller. You finished by playing the game and thinking of ways to improve it.

Q&A

Q. Why use Continuous Dynamic collision detection on the chaos balls? I thought this setting reduced performance.

A. Continuous collision detection can, in fact, reduce performance. In this instance, it is needed, however. The chaos balls are small and fast enough that sometimes they can pass right through the walls.

Q. Why did I need to create a Chaos tag if I never used it?

A. You now have this tag ready for making improvements in the following exercise.

Workshop

Take some time to work through the questions here to ensure that you have a firm grasp of the material.

Quiz

1. How does a *Chaos Ball* player lose the game?
2. What is the positional axis on which all the ball objects are constrained?
3. True or False: The goals in *Chaos Ball* use the method `OnTriggerEnter()` to determine whether an object is the correct ball.
4. Why are some basic features omitted from the game *Chaos Ball*?

Answers

1. This is a trick question. The player cannot lose the game.
2. The y axis
3. True
4. To give you a chance to add them

Exercise

One of the best parts of making games is that you get to make them the way you want. Following a guide can be a good learning experience, but you don't get the satisfaction of making a custom game. In this exercise, you have an opportunity to modify the game a little to make something more unique. Exactly how you change the game is up to you. Here are some suggestions:

- Try adding a button that allows the player to play again whenever the game is completed. (Graphical user interface elements haven't been covered yet, but this feature exists in the *Amazing Racer* game you created in Hour 6, so see if you can figure it out.)
- Try adding a timer so that the player knows how long it took to win.
- Try adding variations of the chaos balls.
- Try adding a chaos goal that you must bounce all the chaos balls into.
- Try changing the size or shape of the player's bumper.
- Try making a complex bumper out of many shapes.