Chapter No. 6
"Playing and Manipulating Sounds"
In this package, you will find:
A Biography of the authors of the book
A preview chapter from the book, Chapter NO.6 "Playing and Manipulating Sounds"
A synopsis of the book’s content
Information on where to buy this book

About the Authors

Matt Smith is senior lecturer in computing at the Institute of Technology Blanchardstown, Dublin, Ireland (www.itb.ie). In 1980 (you do the math) Matt started computer programming (on a ZX80) and has been programming ever since. In 1985, Matt wrote the lyrics, and was a member of the band that played (and sang, sorry about that by the way) the music on the B-side of the audio cassette carrying the computer game Confuzion (wikipedia.org/wiki/Confuzion).

Matt holds a bachelor's degree in Business Computing (Huddersfield University, UK), and as that was a bit boring, he went on to get a masters in Artificial Intelligence (Aberdeen University, Scotland), and a PhD in Computational Musicology (Open University, UK). Having run out of money after 10 years as a full-time student, he began his career as a lecturer and academic. He has been lecturing and researching on programming, artificial intelligence, web development, and interactive multimedia for almost 20 years, holding full-time positions at Winchester University and London's Middlesex University, before moving to his present post in Ireland in 2002. In recent years, Matt has replaced Flash-based 2D multimedia with Unity-based 3D game development and interactive virtual environments subjects for his computing and digital media undergraduates.
To keep himself fit, Matt took up the Korean martial art of Taekwon-Do (he developed and runs his club's website at www.maynoothtkd.com), and a group of his BSc students are now developing a Unity-based Taekwon-Do interactive "tutor" with Microsoft Kinect cameras. Some of his previous Irish-French student team games can be found and played at www. saintgermes.com (thanks for continuing to host these, Guillem!). Matt was one of the two technical experts for a recent multimedia European project for language and cultural student work mobility (vocalproject.eu).

Matt is currently struggling to learn Korean (for his Taekwon-Do), and Irish (since his daughter Charlotte attends an Irish-speaking school and he doesn't believe her translations of her teacher's report cards ...). In 2012, he started taking classical piano lessons again (after a 20-year gap), with a view to sitting exams starting May, 2013.


Thanks to my family for all their support. Thanks also to my students, who continue to challenge and surprise me with their enthusiasm for multimedia and game development.

I would like to dedicate this book to my wife Sinead and children Charlotte and Luke.

For More Information:  
www.packtpub.com/unity-4-x-cookbook/book
Chico Queiroz is a multimedia designer from Rio de Janeiro, Brazil. Chico initiated his career back in 2000, soon after graduating in Communications/Advertising (PUC-Rio), working with advergames and webgames using Flash and Director at LocZ Multimedia. Here he contributed to the design and development of games for clients, such as Volkswagen and Parmalat, along with some independent titles.

Chico has a Master's Degree in Digital Game Design (University for the Creative Arts, UK). His final project was exhibited at events and festivals, such as London Serious Games Showcase and FILE. Chico has also published articles for academic conferences and websites, such as gameology.org, gamasutra.com, and gamecareerguide.com.

He curated and organized an exhibition, held at SBGames 2009, which explored connections between video games and art. SBGames is the annual symposium of the Special Commission of Games and Digital Entertainment of the Computing Brazilian Society.

Chico currently works as a Digital Designer at the Computer Graphics Technology Group (TecGraf), within the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), where he, among other responsibilities, uses Unity to develop interactive presentations and concept prototypes for interactive visualization software. He also works as a lecturer at PUC-Rio, teaching undergraduate Design students 3D modeling and Technology/CG for Games, in which Unity is used as the engine for the students' projects.

I would like to thank my friends, family, co-workers, and all who have made this book possible and have helped me along the way. Special thanks to Stefano Corazza, Anaïs Gragueb, and Oliver Barraza for their fantastic work at Mixamo; Eduardo Thadeu Corseuil, my manager at TecGraf, for giving me the opportunity of using Unity in our interactive projects. Peter Dam and Peter Hohl from TecGraf, and Paul Bourke from the University of Western Australia, for their help and advice on stereo 3D visualization; Aldo Naletto for sharing his knowledge on sound engineering; my students and colleagues at PUC-Rio Art and Design department.

I would like to dedicate this book to my wife Ana and my daughters Alice and Olivia. Thank you for all your love and support.

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
Unity 4.x Cookbook

Game development is a broad and complex task. An interdisciplinary field covering subjects as diverse as Artificial Intelligence, character animation, digital painting, and sound editing. All those areas of knowledge can materialize as the production of hundreds (or thousands!) of multimedia and data assets. A special software application—the game engine—is required to consolidate all of those assets into a single product.

Game engines are specialized pieces of software, which used to belong to an esoteric domain. They were expensive, inflexible, and extremely complicated to use. They were for big studios or hardcore programmers only. Then along came Unity.

Unity represents true democratization of game development. An engine and multimedia editing environment that is user-friendly and versatile. It has free and indie versions and a Pro version that includes even more features. As we write this preface, Unity offers modules capable of publishing games to Windows, Mac, Linux, iOS, Android, XBox 360, Wii U, and PS3; as well as web-based games using the Unity plugins.

Today, Unity is used by a diverse community of developers all around the world. Some are students and hobbyists, but many are commercial organizations ranging from garage developers to international studios, using Unity to make a huge number of games—some you might have already played in one platform or another.

This book provides over 100 Unity game development recipes. Some recipes demonstrate Unity application techniques for multimedia features, including working with animations and using preinstalled package systems. Other recipes develop game components with C# scripts, ranging from working with data structures and data file manipulation, to artificial intelligence algorithms for computer controlled characters.

If you want to develop quality games in an organized and straightforward way, and want to learn how to create useful game components and solve common problems, then both Unity and this book are for you.

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
What This Book Covers

*Chapter 1, Getting Started with Unity 4.x,* is written for those who have just started, or are about to start, using Unity 4.x. It covers software installation, interface concepts, user preferences, and some workflow tips.

*Chapter 2, Using Cameras,* will explain recipes covering techniques for controlling and enhancing your game's camera. This chapter will present interesting solutions to work with both single and multiple cameras.

*Chapter 3, Creating Maps and Materials,* contains recipes that will give you—whether you are a game artist or not—a better understanding on how to use maps and materials in Unity 4.x. It should be a great resource for exercising your image editing skills.

*Chapter 4, Creating GUIs,* is filled with GUI (Graphical User Interface) recipes to help you increase the entertainment and enjoyment of your games through the quality of the interactive visual elements. You'll learn a wide range of GUI techniques, including working with scroll wheels for input, and displaying directional compasses, radars, and graphical inventory icons.

*Chapter 5, Controlling Animations,* demonstrates focusing on character animation, how to take advantage of Unity's new animation system—Mecanim. It covers everything from basic character setup to procedural animation and ragdoll physics.

*Chapter 6, Playing and Manipulating Sounds,* is dedicated to making sound effects and soundtrack music in your game more interesting. It also touches on playback and volume control techniques.

*Chapter 7, Working with External Resource Files and Devices,* throws light on how external data can enhance your game in ways, such as adding renewable content and communicating with websites. External devices, such as the Microsoft Kinect, can totally change the game's interactions. Learn about communicating with external resources and devices in this chapter.

For More Information:

www.packtpub.com/unity-4-x-cookbook/book
Chapter 8, *Working with External Text Files and XML Data*, provides recipes for different methods to work with text files in general, and with XML text data specifically. This chapter is included because XML and other text-based data is common and very useful, both being computer and human readable.

Chapter 9, *Managing Object States and Controlling Their Movements*, relates to the many games that involve moving computer-controlled objects and characters. For many games animation components can be sufficient. However, other games use artificial intelligence for directional logic. This chapter presents a range of such directional recipes, which can lead to games with a richer and more exciting user experience.

Chapter 10, *Improving Games with Extra Features and Optimization*, provides several recipes providing some ideas for adding some extra features to your game (pausing, slow motion, 3D stereography, and securing online games). The rest of the recipes in this chapter provide examples of how to investigate and improve the efficiency and performance of your game's code.

Chapter 11, *Taking Advantage of Unity Pro*, is a concise chapter with interesting uses for some Unity Pro capabilities. It includes recipes for sound, render texture, video texture, and image effects.

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
Playing and Manipulating Sounds

In this chapter, we will cover:

- Matching audio pitch to animation speed
- Adding customizable volume controls
- Simulating a tunnel environment with Reverb Zones
- Preventing the AudioClip from restarting if already playing
- Waiting for audio to finish before auto-destructing an object
- Making a dynamic soundtrack

Introduction

Sound is a very important part of the gaming experience. In fact, we can't stress enough how crucial it is to the player's immersion in a virtual environment. Just think of the engine running in your favorite racing game, the distant urban buzz in a simulator game, or the creeping noises in horror games. Think of how those sounds transport you into the game. This chapter is filled with recipes that, hopefully, will help you implement a better and more efficient sound design to your projects.

Matching audio pitch to animation speed

Many artifacts sound higher in pitch when accelerated and lower when slowed down. Car engines, fan coolers, a Vinyl record player... the list goes on. If you want to simulate this kind of sound effect in an animated object that can have its speed changed dynamically, follow this recipe.

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
Getting ready

For this recipe, you'll need an animated 3D object and an audio clip. Please use the carousel.fbx and carouselSound.wav files, available in the 0423_06_01 folder.

How to do it...

To change the pitch of an audio clip according to the speed of an animated object, please follow these steps:

1. Import the carousel.fbx file into your Unity project.
2. Select the carousel.fbx file in the Project view. Then, in the Inspector view, check its Import Settings. Under Animations, select the Take 001 clip and make sure to check the Loop Pose option. Click the Apply button to save changes:

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
3. Add the carousel to the scene by dragging it from the **Project** view into the **Hierarchy** view.

4. Add a **Directional Light** to the scene through the **Create** drop-down menu on top of the **Hierarchy** view.

5. Import the `carouselSound.wav` audio clip file.

6. Select the **carousel** game object and drag `carouselSound` from the **Project** view into the **Inspector** view, adding it as an audio source for that object.

7. In the **Audio Source** component of the carousel, check the box for the **Loop** option:

8. We need to create a controller for our object. In the **Project** view, click the **Create** button and select **Animator Controller**. Name it `CarouselController`.

For More Information:

www.packtpub.com/unity-4-x-cookbook/book
9. Double click **CarouselController** to open the **Animator** view. Then, right-click the gridded area and select **Create State | Empty** from the contextual menu:

10. Name the new state **spin** and set **Take 001** as its motion in the **Motion** field:

11. From the **Hierarchy** view, select the carousel. Then, in the **Animator** component (in the **Inspector** view), set **CarouselAnimator** as its **Controller** and uncheck the **Apply Root Motion** option:

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
12. In the **Project** window, create a new **C#** script and rename it as `ChangePitch`.

13. Open the script in your editor and replace everything with the following code:

   ```csharp
   using UnityEngine;

   public class ChangePitch : MonoBehaviour{
       public float speed = 0.0f;
       public float minSpeed = 0.0f;
       public float maxSpeed = 2.0f;
       public float animationSoundRatio = 1.0f;
       private Animator animator;

       void Start(){
           animator = GetComponent<Animator>();
       }

       void Update(){
           animator.speed = speed;
           audio.pitch = speed * animationSoundRatio;
       }

       void OnGUI(){
           Rect rect = new Rect(10, 10, 100, 30);
           speed = GUI.HorizontalSlider(rect, speed, minSpeed, maxSpeed);
       }
   }
   ```

14. Save your script and add it as a component to the carousel.

15. Play the scene and change the animation speed, along with the audio pitch, using the slider.

**How it works...**

The idea behind the script and its implementation are actually quite straightforward. It creates a slider from which the user can change the speed of the animator component. Then, it updates the audio pitch based on that number.

**There's more...**

Here is some information on how to fine-tune and customize this recipe.

**Changing the Animation / Sound Ratio parameter**

If you want the audio clip pitch to be either more or less affected by the animation speed, change the value of the **Animation / Sound Ratio** parameter.

---

For More Information:

[www.packtpub.com/unity-4-x-cookbook/book](www.packtpub.com/unity-4-x-cookbook/book)
Adding customizable volume controls

Sound volume adjustment can be a very important feature, especially if your game is a standalone. After all, it can be very frustrating needing to access the operational system volume control. In this recipe, we will create a sound volume control GUI that can be switched from a single volume bar to independent music and effects controls.

Getting ready

For this recipe, you'll need the soundFX.wav and soundtrack.mp3 audio files, available in the 0423_06_02 folder.

How to do it...

To add volume control sliders to your scene, follow these steps:

1. Import the required soundtrack.mp3 audio file.
2. In the Project view, select the soundtrack.mp3 file. Make sure the 3D Sound option of the Audio Importer (in the Inspector view) is unselected. If not, unselect it.
3. Let's now import the soundFX.wav audio clip. This time, we will make sure to leave the 3D Sound option checked.
4. Make sure the First Person Controller prefab is available in your project. You can do that by importing it from Assets | Import Package... | Character Controller.
5. Add the First Person Controller prefab to your scene by dragging it from the Project view to the Hierarchy view. Then, in the Inspector view, reset its position to X: 0, Y: 0, Z: 0, as shown in the following screenshot:

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
6. Delete the original Main Camera from the scene.

7. Expand the First Person Controller object in the Hierarchy view and select the Main Camera child.

8. Drag the soundtrack audio clip file from the Project view to the bottom of the Main Camera Inspector view, adding it as an Audio Source component. Check the Loop option:

9. Using the Create drop-down of the Hierarchy view, add a Plane to your scene.

10. In the Transform component of the Inspector view, change the Plane’s Scale X and Z values to 100. Also, change its position to X: 0, Y: -2, Z: 0. The Plane object will be the ground for our scene:

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
11. Using the **Create** drop-down of the **Hierarchy** view, add a Cube to your scene and rename it to **FXSource**. Set its position to **X:0, Y:0, Z:6**.

12. Drag the **soundFX** clip from the **Project** view to bottom of the **FXSource Inspector** view, adding it as an **Audio Source** component.

13. In the **Audio Source** component of the **FXSource** cube, check the **Loop** option and set **Doppler Level** to **0**.

14. Add a **Directional Light** to the scene.

15. In the **Project** view, create a new C# script and rename it **VolumeControl**.

16. Open the script in your editor and replace everything with the following code:

```csharp
using UnityEngine;

[RequireComponent(typeof(AudioSource))]
public class VolumeControl : MonoBehaviour{
    bool separateSoundtrack = true;
    float minVolume = 0.0f;
    float maxVolume = 1.0f;
    float initialVolume = 1.0f;
    float soundtrackVolume = 1.0f;
    bool displaySliders = false;

    void Start(){
        if (separateSoundtrack){
            audio.ignoreListenerVolume = true;
        }
    }

    void Update(){

    }
}
```

---

For More Information:

[www.packtpub.com/unity-4-x-cookbook/book](http://www.packtpub.com/unity-4-x-cookbook/book)
AudioListener.volume = initialVolume;
if (separateSoundtrack){
  audio.volume = soundtrackVolume;
} else{
  audio.volume = initialVolume;
}
}

void OnGUI(){
  Event e = Event.current;
  if (e.type == EventType.KeyUp && e.keyCode == KeyCode.Escape){
    displaySliders = !displaySliders;
  }
  if (displaySliders){
  if (!separateSoundtrack){
    GUI.Label(new Rect(10, 0, 100, 30), "Volume");
    initialVolume = GUI.HorizontalSlider(new Rect(10, 20, 100, 30), initialVolume, minVolume, maxVolume);
  } else{
    GUI.Label(new Rect(10, 0, 100, 30), "Sound FX");
    initialVolume = GUI.HorizontalSlider(new Rect(10, 20, 100, 30), initialVolume, minVolume, maxVolume);
    GUI.Label(new Rect(10, 40, 100, 30), "Music");
    soundtrackVolume = GUI.HorizontalSlider(new Rect(10, 60, 100, 30), soundtrackVolume, minVolume, maxVolume);
  }
  }
}

17. Save your script and attach it to the Main Camera by dragging it from the Project view to camera game object in the Hierarchy view.

18. Play the scene and hit Esc in your keyboard. You'll see the volume slidebars on the top-left of the game's viewport.

How it works...

By default, every sound in the scene has its volume controlled by the Volume parameter of the camera's Audio Listener component. With our script, we assign the Volume slidebar value to that parameter. Also, we create a separate volume bar for the soundtrack.mp3 file, making it independent from the camera's Audio Listener component.

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
There's more...

Here is some information on how to fine-tune and customize this recipe.

Using a single volume bar
If you need to simplify the volume controls even more, you can use a single volume bar by leaving the Separate Soundtrack option unchecked.

Bypassing the Esc key
If you want your volume bars to be displayed automatically, make sure to tick the checkbox for the parameter named Display Sliders.

See also
- The Making a dynamic soundtrack recipe.
- The Pausing the game recipe.

Simulating a tunnel environment with Reverb Zones

Once you have created your level's geometry, and the scene is looking just the way you want it to, you might want your sound effects to correspond to that look. Sound behaves differently depending on the environment it is projected, so it can be a good idea to make it reverberate accordingly. In this recipe, we will address this acoustic effect by using Reverb Zones.

Getting ready
For this recipe, we have prepared a package containing a basic level named reverbZoneLevel and the signal prefab. The package is in the 0423_06_03 folder.

How to do it...
Follow these steps to simulate the sonic landscape of a tunnel:

1. Import the reverbZones package into your Unity project.
2. In the Project view, open the reverbZoneLevel level, inside the 06_03 ReverbZones folder. This is a basic scene featuring a first-person camera and a tunnel.
3. Now drag signalPrefab from the Project view into the Hierarchy view. That should add a sound-emitting object to the scene. Place it in the center of the tunnel.

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
4. Make five copies of the `signalPrefab` game object and distribute them across the tunnel (leaving a copy just outside each entrance), as shown here:
5. In the **Hierarchy** view, click **Create** to add an **Audio Reverb Zone** to the scene. Now place it in the center of the tunnel.

6. Select the **Reverb Zone** game object. In the **Inspector** view, change the **Reverb Zone** component parameters to these values: **Min Distance**: 3; **Max Distance**: 9; **Preset**: **StoneCorridor**.

7. Play the scene and walk through the tunnel. You should hear the audio reverberate when inside the **Reverb Zone** area.

**How it works...**

Once positioned, the **Audio Reverb Zone** applies an audio filter to all audio sources within its radius.

**There's more...**

Here are more options for you to try.

**Attaching the Audio Reverb Zone component to audio sources**

Instead of creating an **Audio Reverb Zone** game object, you could attach it to the sound emitting object (in our case, **signalPrefab**) as a component by navigating to **Component | Audio | Audio Reverb Zone**. In this case, the **Reverb Zone** would be individually set up around the object.

---

For More Information:  
[www.packtpub.com/unity-4-x-cookbook/book](http://www.packtpub.com/unity-4-x-cookbook/book)
Making your own Reverb settings

Unity comes with several Reverb presets. We have used StoneCorridor, but your scene could ask for something less intense (such as Room) or more radical (such as Psychotic). If those presets still won’t be able to recreate the effect you have in mind, change it to User and edit the parameters as you wish.

Preventing the AudioClip from restarting if already playing

In a game there may be several different events that cause a sound to start playing. If the sound is already playing, then in almost all cases we don't wish to restart the sound. This recipe includes a test, so that an AudioSource component is only sent a Play() message if it is not currently playing.

Getting ready

Try this with any audio clip that is one second or longer in duration.

How to do it...

To prevent an audio clip from restarting, follow these steps:

1. Create an empty game object named AudioObject, and add an audio source component to this object.
2. Drag an audio clip file from the Project view to populate the AudioClip parameter of the AudioSource component of AudioObject.
3. Add the following script class to the Main Camera:

   // file: AvoidSoundRestart.cs
   using UnityEngine;
   public class AvoidSoundRestart : Mono Behaviour{
     public AudioSource audioSource;
     private void OnGUI(){
       string statusMessage = "audio source - not playing";
       if(audioSource.isPlaying )
         statusMessage = "audio source - playing";
       GUI.Label( statusMessage );
   }
Playing and Manipulating Sounds

bool buttonWasclicked = GUILayout.Button("send Play() message");
if( buttonWasclicked )
    PlaySoundIfNotPlaying();
}

private void PlaySoundIfNotPlaying(){
    if( !audioSource.isPlaying )
        audioSource.Play();
}

4. With the Main Camera selected in the Hierarchy view, drag AudioObject into the Inspector for the public AudioSource variable.

How it works...

AudioSource components have a public readable property named isPlaying, which is a Boolean true/false flag indicating if the sound is currently playing. The PlaySoundIfNotPlaying() method includes an if statement ensuring that a Play() message is only sent to the AudioSource component if its isPlaying is false.

See also

- The Waiting for audio to finish before auto-destructing an object recipe.

Waiting for audio to finish before auto-destructing an object

An event may occur (such as an object pickup, or the killing of an enemy) that we wish to notify to the player by playing an audio clip and an associated visual object (such as an explosion particle system, or a temporary object in the location of the event). However, as soon as the clip has finished playing, we want the visual object to be removed from the scene. This recipe provides a simple way to link the ending of an audio clip that's playing with the automatic destruction of its parent GameObject.

Getting ready

Try this with any audio clip that is one second or longer in duration.

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
How to do it...

To wait for the audio to finish before destroying an object, follow these steps:

1. Create an empty game object named `AudioObject`, and add an audio source component to this object.

2. Drag an audio clip file from the `Project` view to populate the `AudioClip` parameter of the `AudioSource` component of `AudioObject`, and deselect the component's `Play On Awake` checkbox.

3. Add the following script class to `AudioObject`:

```csharp
// file: AudioDestructBehaviour.cs
using UnityEngine;
using System.Collections;

public class AudioDestructBehaviour : MonoBehaviour {
    private void Update()
    {
        if( !audio.isPlaying )
            Destroy(gameObject);
    }
}
```

4. Add the following script class to the `Main Camera`:

```csharp
// file: PlayDestroyButtonGUI.cs
using UnityEngine;
using System.Collections;

public class PlayDestroyButtonGUI : MonoBehaviour{
    public AudioDestructBehaviour myAudioDestructObect;

    private void OnGUI(){
        bool playButtonWasClicked = GUILayout.Button("play");
        bool destroyButtonWasClicked = GUILayout.Button("play then destroy");

        if( playButtonWasClicked ){
            myAudioDestructObect.audio.Play();
        }

        if( destroyButtonWasClicked ){
            myAudioDestructObect.audio.Play();
            myAudioDestructObect.enabled = true;
        }
    }
}
```
5. With the **Main Camera** selected in the **Hierarchy** view, drag **AudioObject** into the **Inspector** view for the public **AudioSource** variable myAudioDestructObject.

6. With the **AudioObject** selected in the **Hierarchy** view, disable the scripted component **AutoDestructBehaviour** (uncheck the box by this component).

### How it works...

The game object named **AudioObject** contains an **AudioSource** component, which stores and manages the playing of audio clips. **AudioObject** also contains a scripted component, which is an instance of the **AudioDestructBehaviour** class. When enabled, in every frame this object (via its **Update()** method) tests whether the audio source is not playing (!audio.isPlaying). As soon as the audio is found not to be playing the game object is destroyed. While the audio source is playing, then no action is taken.

The **Main Camera** scripted object **PlayDestroyButtonGUI** offers two buttons to the user, both buttons send a **Play()** message to the audio source component of the audio game object. However, when the **Play then destroy** button is clicked, the scripted component is also enabled. By enabling the scripted object, it means the logic in its **Update()** method will be tested each frame, and as soon as the audio clip has finished playing, then the parent game object will be destroyed.

### See also

- The Preventing the AudioClip from restarting if already playing recipe.

### Making a dynamic soundtrack

Dynamic soundtracks are the ones that change according to what is happening to the player in the game, musically reflecting that place or moment of the character's adventure. In this recipe, we will implement a soundtrack that changes when the player reaches specific targets. Also, we will have the option of fading the sound in and out.

### Getting ready

For this recipe, we have prepared a basic level and some soundtrack audio files in .ogg format. They are contained inside the Unity package named **DynamicSoundtrack**, which can be found in the 0423_06_06 folder.

For More Information:

[www.packtpub.com/unity-4-x-cookbook/book]
How to do it...

To make a dynamic soundtrack, follow these steps:

1. Import the DynamicSoundtrack package into your Unity project. Also, import the _00_main, _01_achievement, and _02_danger_ audio files.

2. Open the level named SoundtrackScene. It should include a basic terrain, a 3rd Person Controller and three spheres named Music Sphere.

3. In the Project view, create a new C# script and name it DynamicSoundtrack.

4. Open the script in your editor and replace everything with the following code:

```csharp
using UnityEngine;
using System.Collections;

public class DynamicSoundtrack : MonoBehaviour{
    public AudioClip[] clips;
    public int startingTrack = 0;
    private int currentTrack;
    private int nextTrack;
    private bool isFadingOut = false;
    private float fadeOutTime = 1.0f;
    private bool isFadingIn = false;
    private float fadeInTime = 1.0f;
    private bool waitSequence = true;
    private bool keepTime = false;
    private float targetVolume = 1.0f;
    private float oldVolume = 0.0f;
    private float fadeOutStart = 0.0f;
    private float fadeInStart = 0.0f;

    void Start(){
        audio.clip = clips[startingTrack];
        audio.Play();
        currentTrack = startingTrack;
    }

    void Update(){
        if (isFadingOut){
            if (audio.volume > 0){
                float elapsOut = Time.time - fadeOutStart;
                float indOut = elapsOut / fadeOutTime;
                audio.volume = oldVolume - (indOut * oldVolume);
            }
        }
    }
}
```

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
Playing and Manipulating Sounds

isFadingOut = false;
StartCoroutine(PlaySoundtrack());
}
}

if (isFadingIn){
    if (audio.volume < targetVolume){
        float elapsIn = Time.time - fadeInStart;
        float indIn = elapsIn / fadeInTime;
        audio.volume = indIn;
    }else{
        audio.volume = targetVolume;
        isFadingIn = false;
    }
}

public void ChangeSoundtrack(int newClip, bool waitForSequence, bool keepPreviousTime, float trackVolume, float fadeIn, float fadeOutPrevious){
    nextTrack = newClip;
    waitSequence = waitForSequence;
    keepTime = keepPreviousTime;
    targetVolume = trackVolume;
    fadeInTime = fadeIn;

    if (newClip != currentTrack){
        currentTrack = newClip;
        if (fadeOutPrevious != 0){
            oldVolume = audio.volume;
            fadeOutStart = Time.time;
            fadeOutTime = fadeOutPrevious;
            isFadingOut = true;
        }else{
            StartCoroutine(PlaySoundtrack());
        }
    }
}
IEnumerator PlaySoundtrack(){
    if (waitSequence)
        yield return new WaitForSeconds(audio.clip.length - ((float)audio.timeSamples / (float)audio.clip.frequency));
    if(fadeInTime !=0){

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
audio.volume = 0;
fadeInStart = Time.time;
isFadingIn = true;
}
float StartingPoint = 0.0f;
if (keepTime)
    StartingPoint = audio.timeSamples;

audio.clip = clips[nextTrack];
audio.timeSamples = Mathf.RoundToInt(StartingPoint);
audio.Play();
}

In case you are wondering why we are using timeSamples instead of time, it's because the former is more accurate when working with compressed audio files. To find out its actual time, we used the expression audio.timeSamples / audio.clip.frequency. As this is not currently documented in Unity's Scripting Reference, we thank audio engineer Aldo Naletto for this tip (and reviewer Peter Bruun for reminding us of using float, for more precision).

5. Save your script and attach it to the Main Camera by dragging it from the Project view to the Main Camera game object in the Hierarchy view.

6. Select the Main Camera and, in the Inspector view, access the Dynamic Soundtrack component and change the Size parameter of the Clips variable to 3. Then, drag the 0_Main, 1_Achievement, and 2_Danger sound files from the Project view into the appropriate slots. Also, type in 0 into the slot named Starting Track:

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
7. Now, access the **Audio Source** component and make sure the **Loop** option is checked:

![Audio Source Component]

8. In the **Project** view, create a new C# script and name it **TriggerSoundtrack**.

9. Open the script in your editor and replace everything with the following code:

```csharp
using UnityEngine;
using System.Collections;

public class TriggerSoundtrack : MonoBehaviour{
    public bool waitForSequence = true;
    public bool keepTimeAndVolume = false;
    public float trackVolume = 1.0f;
    public float fadeIn = 0.0f;
    public float fadeOutPrevious = 0.0f;
    public int clip;
    private DynamicSoundtrack soundtrack;

    void Awake(){
        soundtrack = Camera.main.GetComponent<DynamicSoundtrack>();
    }

    void OnTriggerEnter(Collider other){
        if (other.gameObject.CompareTag("Player"))
            soundtrack.ChangeSoundtrack(clip, waitForSequence, keepTimeAndVolume, trackVolume, fadeIn, fadeOutPrevious);
    }
}
```

---

For More Information:  
[www.packtpub.com/unity-4-x-cookbook/book](http://www.packtpub.com/unity-4-x-cookbook/book)
10. Save your script and attach it to each one of the music spheres by dragging it from the Project view to the Main Camera game object in the Hierarchy view.

11. Select each Music Sphere object and, in the Inspector view, change the Trigger Soundtrack parameters, as shown in the following screenshot:

```
Music Sphere 1

Script: DynamicSoundtrack
Wait For Sequence: Off
Keep Time And Volume: On
Track Volume: 1
Fade In: 0
Fade Out Previous: 0
Clip: 2

Music Sphere 2

Script: DynamicSoundtrack
Wait For Sequence: Off
Keep Time And Volume: On
Track Volume: 1
Fade In: 2
Fade Out Previous: 2
Clip: 1

Music Sphere 3

Script: DynamicSoundtrack
Wait For Sequence: Off
Keep Time And Volume: On
Track Volume: 1
Fade In: 4
Fade Out Previous: 4
Clip: 0
```

12. Play your scene and direct the character towards each Music Sphere object. The background music will change accordingly.

**How it works...**

We have created two different scripts. The one attached to the Main Camera, DynamicSoundtrack, is responsible for keeping a list of the audio files that make up the entire soundtrack for the level. It also contains all of the functions that control the audio playback, volume transition, and so on. The second one, TriggerSoundtrack, is attached to the Music Sphere objects and triggers soundtrack changes based on the preferences expressed in that component's parameters. They are:

- **Wait For Sequence**: Leave this option checked if you want to wait until the end of the previous audio clip before playing the new part.

Playing and Manipulating Sounds

- **Keep Time And Volume**: Leave it checked to start a new audio clip from the same point where the previous clip was at. This also keeps the volume level from the previous clip.
- **Track Volume**: The volume for the new audio clip (from 0.0 to 1.0).
- **Fade In**: The amount of time in seconds it will take for the new audio clip's volume to fade in.
- **Fade Out Previous**: The amount of time in seconds it will take for the previous audio clip's volume to fade out.

There's more...

Here is some information on how to fine-tune and customize this recipe.

Hiding the triggers

If having milestone objects as triggers feels too obvious for you and your players, you can always make it invisible by disabling the Mesh Renderer component.

Dealing with audio file formats and compression rates

To avoid loss of audio quality, you should import your sound clips using the appropriate file format, depending on your target platform. If you are not sure which format to use, please check out Unity's documentation on this subject at http://docs.unity3d.com/Documentation/Manual/AudioFiles.html.

Using 2D sound

To make the sound volume and balance independent of the audio source position, make sure your audio clip is not set up as 3D Sound (you can check it out at Import Settings in the Inspector view, by selecting the file in the Project view).

For More Information:
www.packtpub.com/unity-4-x-cookbook/book
Where to buy this book

You can buy Unity 4.x Cookbook from the Packt Publishing website:

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.