There are still a few things that the framework must handle before we can create this game. These additions include:

- Sound
- Collision detection

By the end of the chapter you will have a good understanding of how this game was built using the framework and you will have the ability to continue and improve it. In this chapter, we will cover:

- Implementing sound
- Creating game-specific object classes
- Shooting and detecting bullets
- Creating different enemy types
- Developing a game

Using the SDL_mixer extension for sound

The SDL_mixer extension has its own Mercurial repository that can be used to grab the latest source for the extension. It is located at http://hg.libsdl.org/SDL_mixer. The TortoiseHg application can again be used to clone the extension's Mercurial repository. Follow these steps to build the library:

1. Open up TortoiseHg and press *CTRL+SHIFT+N* to start cloning a new repository.
2. Type http://hg.libsdl.org/SDL_mixer into the source box.
3. The **Destination** will be `C:\SDL2_mixer`.
4. Hit **Clone** and wait for completion.
5. Navigate to `C:\SDL2_mixer\VisualC\` and open `SDL_mixer.vcproj` in Visual Studio 2010.
6. As long as the x64 folder outlined in Chapter 2, *Drawing in SDL* was created, the project will convert with no issues.
7. We are going to build the library without MP3 support as we are not going to need it, and also it does not work particularly well with SDL 2.0.
8. Add `MP3_MUSIC_DISABLED` to the **Preprocessor Definitions** in the project properties, which can be found by navigating to **C/C++** | **Preprocessor**, and build as per the `SDL_image` instructions in Chapter 2, *Drawing in SDL*.

Creating the SoundManager class

The game created in this chapter will not need any advanced sound manipulation, meaning the `SoundManager` class is quite basic. The class has only been tested using the `.ogg` files for music and the `.wav` files for sound effects. Here is the header file:

```
enum sound_type
{
  SOUND_MUSIC = 0,
  SOUND_SFX = 1
};

class SoundManager
{
public:

  static SoundManager* Instance()
  {
    if(s_pInstance == 0)
    {
      s_pInstance = newSoundManager();
      return s_pInstance;
    }
    return s_pInstance;
  }

  bool load(std::string fileName, std::string id, sound_type
type);

  void playSound(std::string id, int loop);
  void playMusic(std::string id, int loop);

  private:
```

```
  static SoundManager* s_pInstance;

  std::map<std::string, Mix_Chunk*> m_sfxs;
  std::map<std::string, Mix_Music*> m_music;

  SoundManager();
  ~SoundManager();

  SoundManager(const SoundManager&);
  SoundManager &operator=(const SoundManager&);
};

typedef SoundManager TheSoundManager;
```

The `SoundManager` class is a singleton; this makes sense because there should only be one place that the sounds are stored and it should be accessible from anywhere in the game. Before sound can be used, `Mix_OpenAudio` must be called to set up the audio for the game. `Mix_OpenAudio` takes the following parameters:

```
(int frequency, Uint16 format, int channels, int chunksize)
```

This is done in the `SoundManager`'s constructor with values that will work well for most games.

```
SoundManager::SoundManager()
{
  Mix_OpenAudio(22050, AUDIO_S16, 2, 4096);
}
```

The `SoundManager` class stores sounds in two different `std::map` containers:

```
std::map<std::string, Mix_Chunk*> m_sfxs;
std::map<std::string, Mix_Music*> m_music;
```

These maps store pointers to one of two different types used by `SDL_mixer` (`Mix_Chunk*` and `Mix_Music*`), keyed using strings. The `Mix_Chunk*` types are used for sound effects and the `Mix_Music*` types are of course used for music. When loading a music file or a sound effect into `SoundManager`, we pass in the type of sound we are loading as an `enum` called `sound_type`.

```
bool load(std::string fileName, std::string id, sound_type
type);
```

This type is then used to decide which `std::map` to add the loaded sound to and also which `load` function to use from `SDL_mixer`. The `load` function is defined in

SoundManager.cpp.

```
bool SoundManager::load(std::string fileName, std::string id,
sound_type type)
{
  if(type == SOUND_MUSIC)
  {
    Mix_Music* pMusic = Mix_LoadMUS(fileName.c_str());

    if(pMusic == 0)
    {
      std::cout << "Could not load music: ERROR - "
      << Mix_GetError() << std::endl;
      return false;
    }

    m_music[id] = pMusic;
    return true;
  }
  else if(type == SOUND_SFX)
  {
    Mix_Chunk* pChunk = Mix_LoadWAV(fileName.c_str());
    if(pChunk == 0)
    {
      std::cout << "Could not load SFX: ERROR - "
      << Mix_GetError() << std::endl;

      return false;
    }

    m_sfxs[id] = pChunk;
    return true;
  }
  return false;
}
```

Once a sound has been loaded it can be played using the **playSound** or **playMusic** functions:

```
void playSound(std::string id, int loop);
void playMusic(std::string id, int loop);
```

Both of these functions take the ID of the sound to be played and the amount of times that it is to be looped. Both functions are very similar.

```
void SoundManager::playMusic(std::string id, int loop)
{
  Mix_PlayMusic(m_music[id], loop);
}

void SoundManager::playSound(std::string id, int loop)
```

```
{
  Mix_PlayChannel(-1, m_sfxs[id], loop);
}
```

One difference between `Mix_PlayMusic` and `Mix_PlayChannel` is that the latter takes an `int` as the first parameter; this is the channel that the sound is to be played on. A value of **-1** (as seen in the preceding code) tells `SDL_mixer` to play the sound on any available channel.

Finally, when the `SoundManager` class is destroyed, it will call `Mix_CloseAudio`:

```
SoundManager::~SoundManager()
{
  Mix_CloseAudio();
}
```

And that's it for the `SoundManager` class.