Dr. Edward Lavieri

Learning AWS Lumberyard Game Development

Create stunning 3D multiplayer games with integrated cloud-based features

Packt

Learning AWS Lumberyard Game Development

Create stunning 3D multiplayer games with integrated cloud-based features

Dr. Edward Lavieri

BIRMINGHAM - MUMBAI

Learning AWS Lumberyard Game Development

First published: October 2016

Production reference: 1211016

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.
ISBN 978-1-78646-086-8

www.packtpub.com

Credits

Author

Dr.Edward Lavieri

Reviewer

Randy Lutcavich

Commissioning Editor

Ashwin Nair

Acquisition Editor

Anurag Banerjee

Content Development Editor

Sachin Karnani

Technical Editor

Sachit Bedi

Copy Editor

Safis Editing

Project Coordinator

Ritika Manoj

Proofreader

Safis Editing

Indexer

Tejal Daruwale Soni

Graphics

Abhinash Sahu

Production Coordinator

Melwyn Dsa

About the Author

Dr. Edward Lavieri is a veteran game designer and developer with a strong academic background. He earned a doctorate in computer science from Colorado Technical University and three Master of Science degrees in Management Information Systems (Bowie State University), Education – Instructional Design (Capella University), and Operations Management (University of Arkansas), demonstrating his passion for academic pursuits. He has developed and taught computer-related courses since 2002. Edward retired from the U.S. Navy after 25 years as an Intelligence Specialist and Command Master Chief.

Edward has authored *Adaptive Learning for Educational Game Design*, *Getting Started with Unity 5*, *LiveCode Mobile Development Hotshot*, *LiveCode Mobile Development Cookbook*, *Software Consulting: A Revolutionary Approach,* and was the technical editor of the *Excel Formulas and Functions for Dummies* book. He has also authored numerous computer science and information systems college courses.

To IBB, my ride or die.

About the Reviewer

Randy Lutcavich is a software engineer focused on leveling up his skills in mobile and gaming technology. As the cofounder of WiNF Studios LLC, he creates games for all. You can follow him on Twitter at `@WiNF_Randy`.

www.PacktPub.com

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www.packtpub.com/mapt`

Get the most in-demand software skills with Mapt. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Table of Contents

Preface

As you can guess from the title of the book, this book is designed to introduce game developers to Lumberyard. Lumberyard is a new, open source 3D game engine that provides game developers with the ability to create live multiplayer games with the integration of key Amazon Web Services.

This book teaches the reader how to use Lumberyard to create a multiplayer 3D game with cloud computing and storage and with Twitch integration for its user engagement.

This book will start with an introduction to Lumberyard and an overview of its capabilities and integration options. Once the game engine is installed, the book guides the reader through the creation of an immersive game world with characters. Animations and audio will be added to help bring the game to life. External interactions will be explored to support live multiplayer game play, data storage, user engagement, and backend support.

What this book covers

`Chapter 1`, *Welcome to the Lumberyard*, will give you an initial look at Lumberyard and why it is unique among other game engines. We install the game engine and explore the user interface.

`Chapter 2`, *Planning Your Game in the Lumberyard,* will look into Lumberyard's beta release history and explore how that impacts your development efforts. We will also preview game functionality that we will create in subsequent chapters, look at game design for Lumberyard games, and explore how to plan the development process.

`Chapter 3`, *Constructing an Immersive 3D Game World*, will help you create a game world with trees, a river, hills, mountain, light sources, and shadows. In order to test our game, we will create a player-character and a camera.

`Chapter 4`, *Creating 3D Characters*, will help you to add 3D characters to a game. We also explore Geppetto and its user interface.

`Chapter 5`, *Animating Your Characters*, will examine the process of animating our game characters using Mannequin, Lumberyard's animation tool. This chapter covers Mannequin's user interface and its functionality.

`Chapter 6`, *Creating Gameplay*, will help in making your game interactive. In order to do, we need to create gameplay components. That is the focus of this chapter along with an introduction to Flow Graphs and the Flow Graph User Interface.

`Chapter 7`, *Creating Multiplayer Gameplay*, will examine the requirements of creating a multiplayer game in Lumberyard. We will also explore Amazon GameLift.

`Chapter 8`, *Bringing Your Game to Life with Audio and Sound Effects*, will explore Lumberyard's Audio System**.** We will look at the complexity of Lumberyard audio and examine the components of the Lumberyard Audio System.

`Chapter 9`, *Employing Cloud Computing and Storage*, further explores Amazon Web Services and reviews two additional Web Services: Cloud Canvas and Amazon Simple Storage Service**.**

`Chapter 10`, *Engaging With Users Using Twitch*, will take a singular look at Twitch, the Amazon Web Service that allows people to watch live game streaming. You will learn how to implement Twitch functionality for in-game user interactions.

`Chapter 11`, *Providing Your Game to the World*, will provide you with an overview of the steps necessary to publish your game once it is completed. Specifically, we will look at game builds, how to test them, debug them, and release them.

`Chapter 12`, *Stretching Your Lumberyard Wings*, will explore various possibilities with Lumberyard, beyond the basics. We will explore concepts, such as Virtual Reality (VR), the Waf Build System, Lumberyard's cinematics System, System Streaming, and Memory Handling. We will also explore two additional Amazon Web Services (Simple Query Service and Simple Notification Service).

What you need for this book

This book is intended for use alongside a computer running Lumberyard. The Lumberyard Editor is therefore required to fully realize the benefits of this book. Lumberyard can run on a PC with the following minimum system requirements:

- Windows 7 (64-bit) or Windows 10 (64-bit)
- 8 GB RAM
- 60 GB Hard Disk
- 3 GHz quad-core processor
- DirectX 11 (DX11) compatible video card with at least 2 GB of video RAM (VRAM)

At the time of this book's release, Lumberyard beta 1.5 was the most current version available. There might be slight differences in user interface components between the illustrations in this book and the actual Lumberyard interface.

Who this book is for

This book caters to current and future game developers who have an interest in creating immersive, high-quality 3D games with live, multiplayer features. The book is written with the assumption that the reader will have some knowledge of a game design and software development. Experience with C++ is beneficial, but not required.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "There are five areas of the **Mannequin** interface."

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "These files have a `*.i_caf` file extension."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book-what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Learning-AWS-Lumberyard-Game-Development`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `http://www.packtpub.com/sites/default/files/downloads/LearningAWSLumberyardGameDevelopment_ColorImages.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Welcome to the Lumberyard

The purpose of this chapter is to provide you with a brief overview of Lumberyard, what it is capable of, and what you can do with it. We'll start with a brief discussion of where Lumberyard fits into the game engine landscape. Our initial look at Lumberyard will include system requirements and how it integrates with the cloud for computing and storage. You'll be provided with step-by-step instructions for the download and installation process. Lastly, a tour of the Lumberyard interface will be provided.

In this chapter, you will:

- Understand Lumberyard
- Become familiar with Lumberyard's system requirements
- Download and install Lumberyard
- Understand the Lumberyard **Setup Assistant**
- Download and install required software, SDKs, and plugins
- Become familiar with the Lumberyard Editor and the user interface
- Start a new Lumberyard project

What is Lumberyard?

Lumberyard is a free 3D game engine that has, in addition to typical 3D game engine capabilities, an impressive set of unique qualities. Most impressively, Lumberyard integrates with **Amazon Web Services** (**AWS**) for cloud computing and storage. You will learn about AWS in `Chapter 9`, *Employing Cloud Computing and Storage*. Lumberyard, also referred to as Amazon Lumberyard, integrates with *Twitch* to facilitate in-game engagement with fans. We'll cover Twitch in `Chapter 10`, *Engaging With Users Using Twitch*.

Another component that makes Lumberyard unique among other game engines is the tremendous support for multiplayer games. As you'll see in `Chapter 7`, *Creating Multiplayer Gameplay*, the use of *Amazon GameLift* empowers developers to instantiate multiplayer game sessions with relative ease.

Lumberyard is presented as a game engine intended for creating cross-platform AAA games. There are two important components of that statement. First, *cross-platform* refers to, in the case of Lumberyard, the ability to develop games for PC/Windows, PlayStation 4, and Xbox One. At the time of this book's publication, additional support for Mac OS, iOS, and Android devices was being worked on. There is no doubt that these additional platforms will be supported soon. The second component of the earlier statement is *AAA games*. A **triple-A** (**AAA**) game is like a top-grossing movie, one that had a tremendous budget, was extensively advertised, and wildly successful. If you can think of a console game (for Xbox One and/or PlayStation 4) that is advertised on national television, it is a sign the title is a AAA game.

Now that this AAA game engine is available for free, it is likely that more than just AAA games will be developed using Lumberyard. This is an exciting time to be a game developer.

More specifically, Amazon hopes that Lumberyard will be used to develop multiplayer online games that use AWS for cloud computing and storage, and that integrate with Twitch for user engagement. The engine is free, but AWS usage is not. Specifics on this issue will be covered `Chapter 9`, *Employing Cloud Computing and Storage*. Don't worry, you can create single-player games with Lumberyard as well.

System requirements

Amazon recommends a system with the following specifications for developing games with Lumberyard:

- PC running a 64-bit version of Windows 7 or Windows 10
- At least 8 GB RAM
- A minimum of 60 GB hard disk storage
- A 3 GHz or greater quad-core processor
- A DirectX 11 (DX11) compatible video card with at least 2 GB of video RAM (VRAM)

As mentioned above, currently, there is no support for running Lumberyard on a Mac OS or Linux computer.

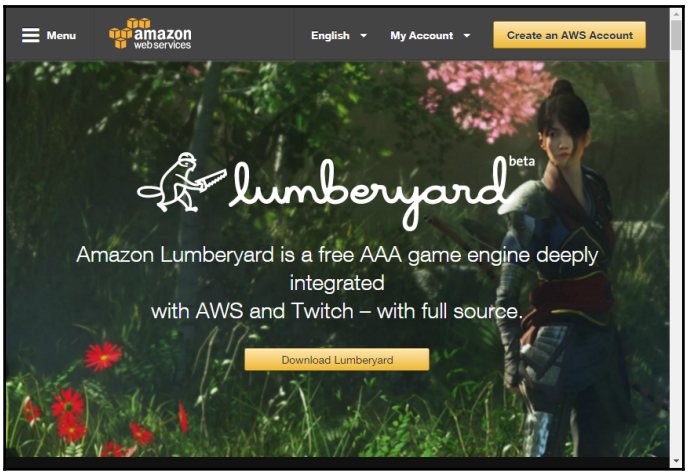
The game engine is a very large and complex software suite. You should take the system requirements seriously and, if at all possible, exceed the minimum requirements.

Downloading and installing Lumberyard

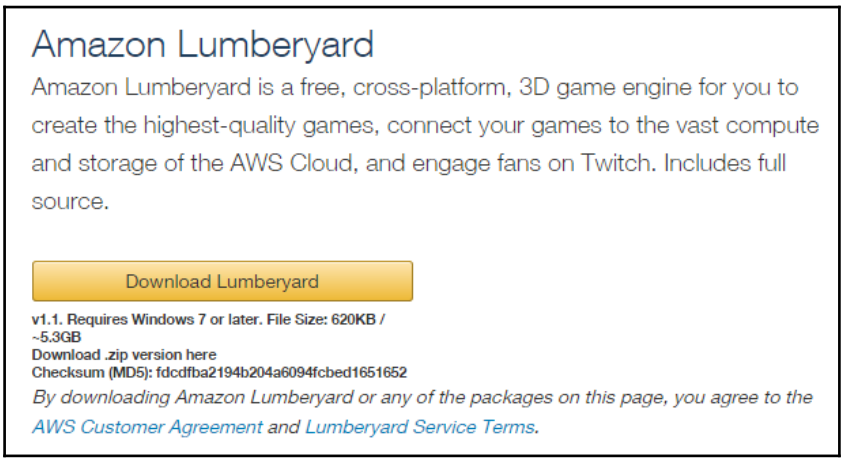
The Internet is a dynamic medium and some links are subject to change after this book's publication date. If a link does not work, you can search for the new web page using an Internet browser. Some of the images, buttons, and other graphical references might be different from what is presented here.

The following steps will guide you through the download and installation processes for Lumberyard. Before following these steps, be sure your system meets the minimum requirements listed in the previous section:

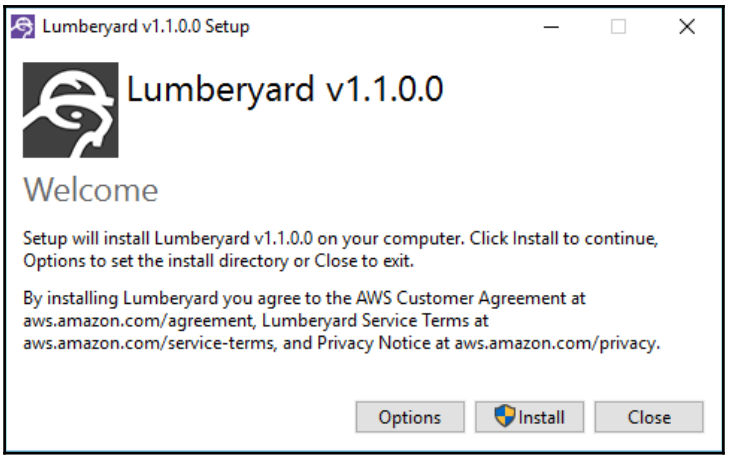
1. Open your Internet browser and navigate to `https://aws.amazon.com/lumberyard`.
2. Find the **Download Lumberyard** button, as shown in the following screenshot, centered and towards the bottom of the screenshot, and click it. This will take you to the `https://aws.amazon.com/lumberyard/downloads/` page:



3. On the downloads page, click the **Download Lumberyard** button. You will be reminded that downloading the game engine indicates you agree to the **AWS Customer Agreement** and **Lumberyard Server Terms**. As shown in the following screenshot, there are links to both of those legal documents beneath the download button:

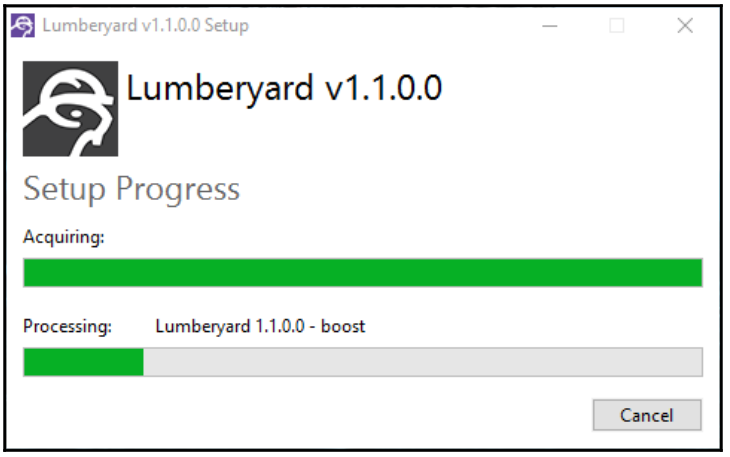


4. Shortly after clicking the **Download Lumberyard** button, you will see that the Lumberyard Installer (filename `LumberyardInstaller1.1.0.0.exe`) was downloaded. Your version number might be slightly different, and that is okay. Double-click the installer to run it.
5. The installer's interface should now be present (refer to the following screenshot). We'll use the default installation directory (`C:\Amazon\Lumberyard`). Click the **Install** button:

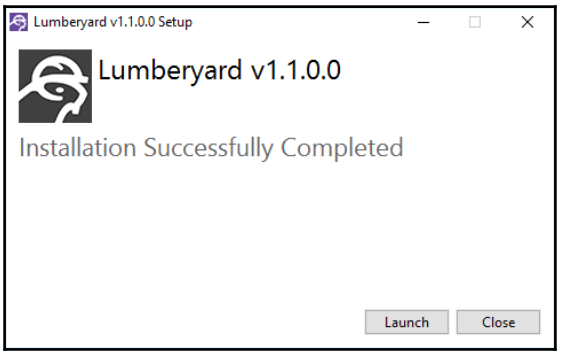


If you see **Modify Setup** when you run the installer, it indicates you have already installed Lumberyard.

6. The installer will now download and install Lumberyard on your computer. The game engine, installed, is approximately 14 GB, so the setup process can be lengthy, even with a lot of bandwidth. You've already ensured there is enough disk space, so now is a great time to grab a cup of coffee. Now you can sit back and monitor the process:



7. When the process is completed, you will be presented with the screen shown in the following screenshot . Click the **Launch** button to run Lumberyard for your first time. This is going to be an exciting journey:

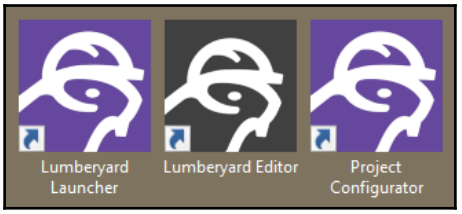


If you have a firewall running on your computer, you might be prompted to grant access for Lumberyard to make changes to your computer.

Launching Lumberyard

You can launch Lumberyard in one of three ways. First, you can click the **Launch** button immediately after installation, as illustrated in the previous section. You can also navigate to the location on your hard drive where you have Lumberyard installed, for example, `C:/Amazon/Lumberyard/1.1.0.0/dev`.

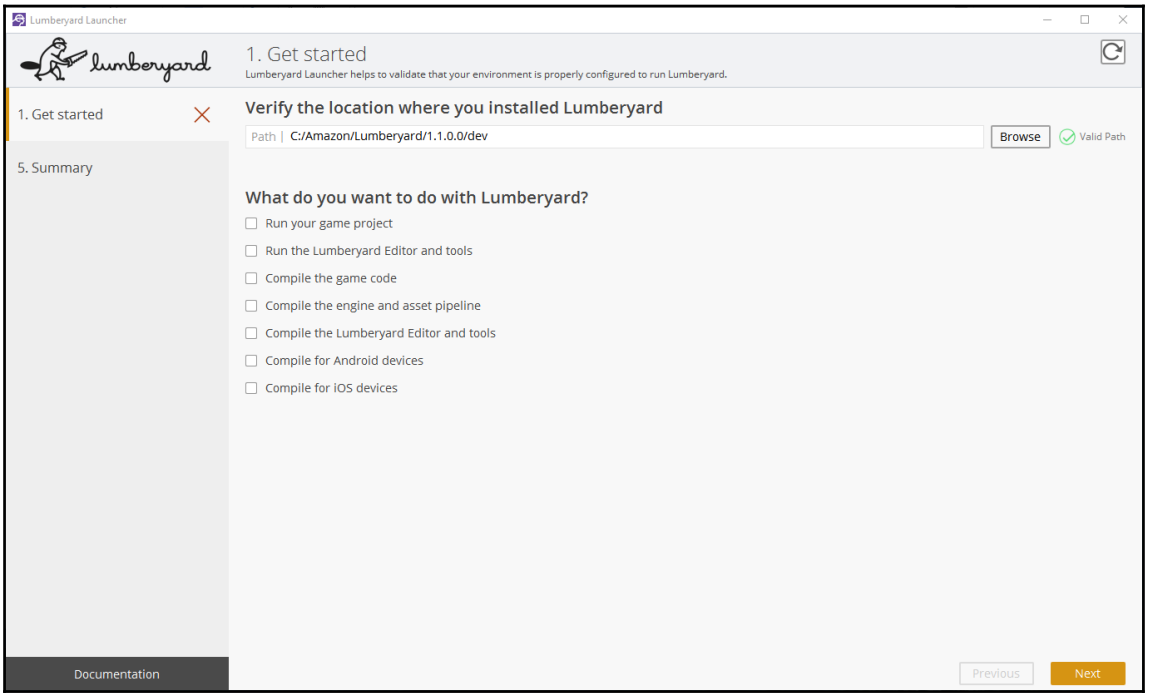
You can also double-click the **Lumberyard Launcher** icon on your computer's desktop. As part of the installation process, you will have three icons related to Lumberyard added to your desktop. We will use the **Lumberyard Launcher** icon now and address the **Lumberyard Editor** and **Project Configurator** icons later:



The names and look of the aforementioned icons might be different on your system, depending on what version of Lumberyard you have installed. For example, the Lumberyard Launcher icon was replaced by/renamed to *Setup Assistant* in v1.2.0.0.

At this point, we want to ensure our installation is complete, including installing any additional software, SDKs, and plugins. The following steps will guide you through the process:

1. After launching the **Lumberyard Launcher**, or **Setup Assistant** depending on your version of the game engine, you are greeted with the **Get started** page. As you can see in the following screenshot, there are several things you can do right from this page:



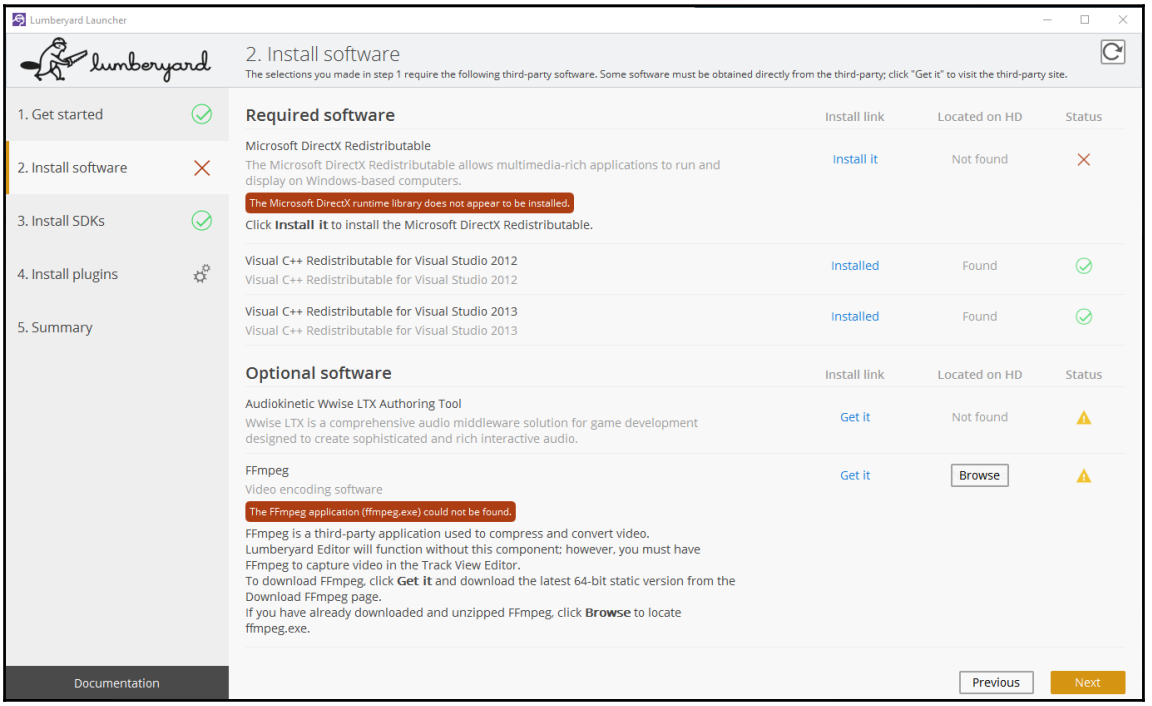
2. On the **Get started** page, there is a link to **Documentation** in the lower-left corner of the screen. More importantly, you are presented with the opportunity to verify your Lumberyard installation location. If you do not see the 

indicator to the right of the **Browse** button, use that button to point the **Launcher**, or **Setup Assistant**, to your installation location.

3. Check the **Run the Lumberyard Editor and tools** checkbox. This will tell the **Launcher**, or **Setup Assistant**, what you want to accomplish and generate additional steps for you to follow.

Depending on your version of Lumberyard, you might need to uncheck the **Run your game project** option.

4. Click the **Next** button in the lower-right corner of the screen. This will result in the **Install software** screen being displayed. As shown in the following screenshot, you might have additional software to install:



5. Using the links provided, install all software listed under the **Required software** heading.
6. You can decide if you want to install the software components listed under the **Optional software** heading. If you do not do this now, you can do it later.
7. Once you have all the desired software installed, select the **Next** button. This will display the **Install SDKs** screen.

8. As appropriate for your needs, install any SDKs that are listed but not already installed. Follow the on-screen guidance. You can always come back to this as your needs change.
9. Once you have all the required SDKs installed, select the **Next** button. This will generate a list of plugins.
10. Install any desired plugins by following the on-screen instructions.
11. Install any additional plugins you want that are listed under the **Available content creation plugins** header.

You will notice that most of the external plugins do not have associated URLs. You will need to install the software manually. You can use the refresh button in the upper-right corner of the screen once your software has been installed. This is a great way to verify that Lumberyard can locate the software.

12. Once you have installed everything you wanted on the **Install plugins** page, click the **Next** button. This will present you with the **Summary** page. Here you can review software, SDKs, and plugins that you still might consider installing. You should see on-screen text indicating that all required software has been installed.
13. On the **Summary** page, click the **Launch Lumberyard Editor** button. The first time you launch the editor, it can take several minutes to load. Lumberyard will perform a lot of housekeeping to ensure your development environment is set up correctly.

If you have a firewall running on your computer, you might need to grant access to the `AssetProcessor_tmp.exe` and `Editor.exe` executable files.

14. During the initial launch process, you will be presented with a **Welcome to Lumberyard** dialog window. Here you will need to enter, or create and enter, your Amazon or AWS account. You can even create a new Amazon account specifically for Lumberyard. This is highly recommended as it will help you segment your dealings with Amazon.

Due to the dynamic nature of the Internet and Amazon's services, your AWS experience might differ slightly from what is presented in this chapter.

15. Create your Amazon account by following the on-screen instructions. This is a free account.

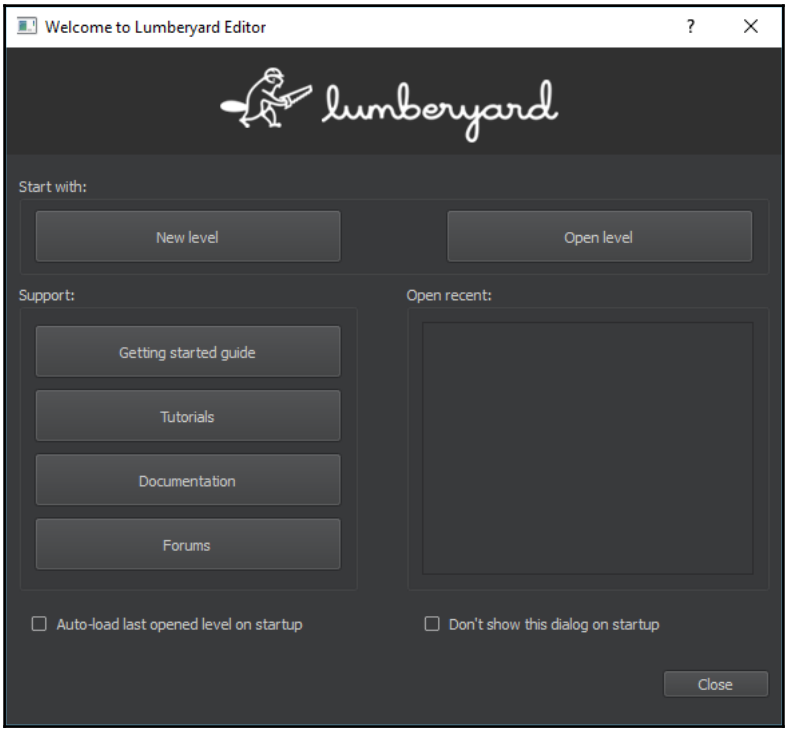
16. After creating your Amazon account and logging in, the Lumberyard Editor will open. Additional guidance is provided in the next section.

Introducing the Lumberyard Editor

The Lumberyard Editor is where you will spend most of your development time. This section will provide an overview of the major components of the **Lumberyard Editor** user interface. Additional details on components of the **Editor** are provided in the later chapters, when the functionality is first introduced.

The Welcome screen

Before you use the **Editor**, at least the first time you launch it, you will be presented with the **Welcome to Lumberyard Editor** dialog window, as shown in the following screenshot:



There are four content areas on the **Welcome** screen and two decisions. The first content area is **Start with**, which allows you to start a new level or open a level you previously created.

Levels are components of games and games are created by developing multiple linked game levels. The **Lumberyard Editor** allows you to create and edit game levels.

The second section on the **Welcome** screen contains four support links, each accessible via a button:

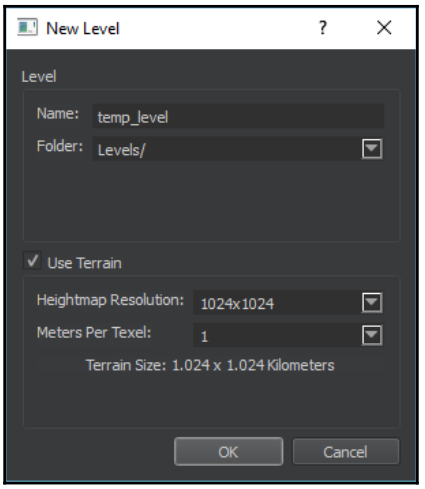
- Getting started guide (opens `https://docs.aws.amazon.com/lumberyard/latest/gettingstartedguide/intro.html` in your browser)
- Tutorials (opens `https://gamedev.amazon.com/forums/tutorials` in your browser)
- Documentation (opens `https://docs.aws.amazon.com/lumberyard/latest/userguide/lumberyard-intro.html` in your browser)
- Forums (opens `https://gamedev.amazon.com/forums/index.html` in your browser)

There is also an **Open recent**: section to the **Welcome** dialog window. This area will list levels that you have previously saved. This gives you quick access to your recent work.

The final content area of the **Welcome** dialog window contains two checkboxes. By selecting or de-selecting the boxes, you can determine whether your last opened level will be automatically loaded when the editor is launched and whether you want to suppress the **Welcome** dialog window on future launches of the editor.

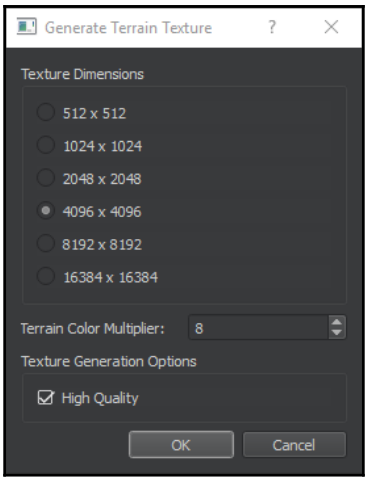
Creating a new level

To review the editor's user interface, we will select **New level** from the **Welcome** dialog window. This generates a **New Level** dialog window. Here, we can name our level and indicate where to store the files associated with the level:



If terrain is being used, you can assign the resolution of the **Heightmap** and how many **Meters Per Texel**. We will look at the terrain settings later, in `Chapter 3`, *Constructing an Immersive 3D Game World*. For now, we will change the level's name to **test_level** and click the **OK** button. We are accepting the default location for new levels; you can change this if needed.

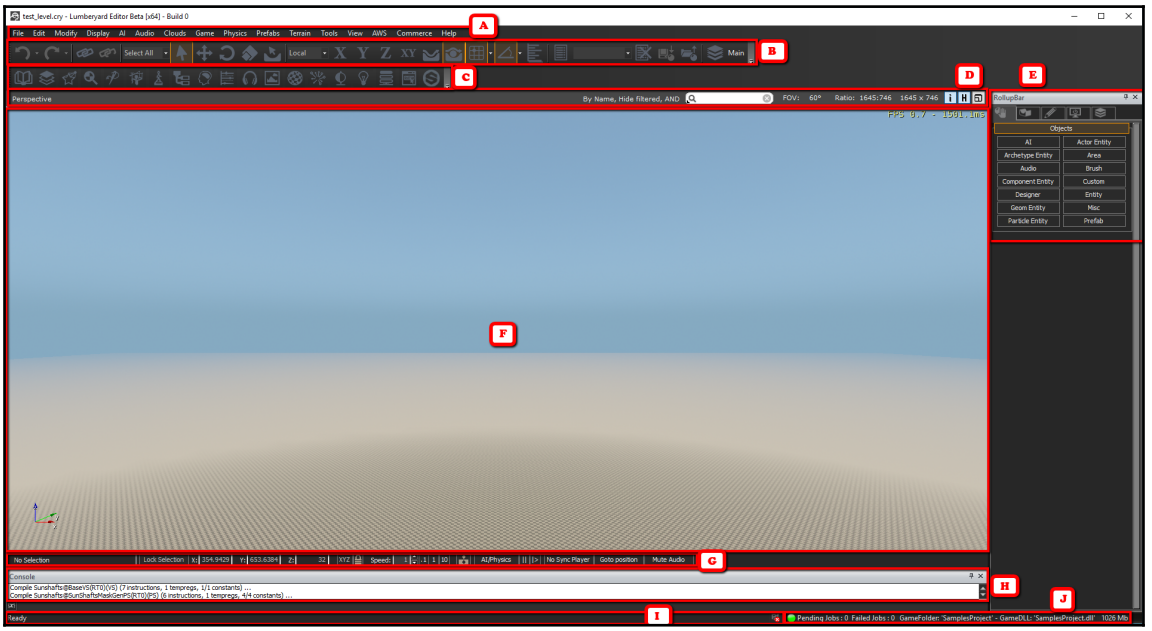
The **Generate Terrain Texture** dialog window appears next and is shown in the following screenshot. As you can see, there are several decisions we need to make here. At this point, we simply want to review user interface of the **Editor**. We'll take an in-depth look at the terrain options in `Chapter 3`, *Constructing an Immersive 3D Game World*. For now, simply click the **OK** button to accept the default settings:



In the next section, we will review user interface of the **Lumberyard Editor**.

Editor user interface – overview

The main interface of **Lumberyard Editor** is comprised of 10 areas, or workspace components. The following screenshot illustrates each of these areas:



Each area of the user interface provides you with access to specific functions. The individual areas are described in the next 10 subsections with reference to the previous screenshot. You will learn more about each area and the supported functionality as we build our first game throughout the remainder of this book.

Pull-down main menu (area A)

This area consists of a menu bar at the top of the interface. Each menu label, listed as follows, presents the user with multiple options and functions:

- **File** – You can open, close, and export projects.
- **Edit** – This menu is contextual and supports grouping, visibility, and more.
- **Modify** – Here you have access to make modifications to your game objects.
- **Display** – You can change and configure how your level is viewed in the viewport.
- **AI** – Access to Lumberyard's artificial intelligence functionality.
- **Audio** – You can refresh audio and stop all sounds.
- **Clouds** – You can create, open, close, and destroy clouds.
- **Game** – This is a set of tools relevant to your game to include enabling physics and AI, edit equipment packs, and more.
- **Physics** – With this menu set, you can get and reset physics states and simulate objects.
- **Prefabs** – Tools to create, edit, and manage prefabs.

A prefab is a group of predefined assets. Using prefabs can speed up the content creation process.

- **Terrain** – Access to terrain-related creation and editing.
- **Tools** – A plethora of tools for scripts, textures, shaders, terrain, geometry, and more.
- **View** – This menu gives you access to tailor your layout and open/close user interface components.
- **AWS** – Access to AWS to include cloud computing and storage, GameLift, and more. This is where you will gain access to your AWS Profile.
- **Commerce** – Web links to the Amazon developer portal and Merch by Amazon.
- **Help** – Access to tutorials, the getting started guide, documentation, and more.

So far, we have only created a test level. You should feel free to explore the menu items to become more familiar with them.

Toolbars (areas B and C)

The default view for the editor includes two rows of toolbars containing icons for commonly used functions and features. You are able to add, remove, and relocate these icons. You can even move the toolbars. This area is highly customizable. It is recommended that you leave the toolbars as is until you become more familiar with the interface and developing with Lumberyard.

Viewport header (area D)

This is an information header bar that pertains to the perspective viewport (area **F**). There is also a search bar to help you find objects. This is especially useful in complex game worlds where selecting a very specific object could be challenging.

Rollup bar (area E)

The rollup bar provides categorized access to a host of game components and functionality. The features are categorized into tabs for **Objects**, **Terrain**, **Modeling**, **Display**, and **Layers**. You will gain exposure to these starting in `Chapter 3`, *Constructing an Immersive 3D Game World*.

The **Modeling** tab is not present in every version of Lumberyard, so if you do not have that tab, that is okay. Once Lumberyard is out of beta, more information will be available about the full game engine.

Perspective viewport (area F)

This viewport provides a visual representation of your game level in 3D.

Viewport controls (area G)

This area is located directly below the viewport. When an object is selected, the viewport controls allow you to modify *X*, *Y*, and *Z* axis values, lock and unlock objects, control speed, and more.

Console (area H)

The console provides systems information, output, game data, and input, as appropriate. It can be a key component when debugging.

Status footer (areas I and J)

The last line of the interface is the status footer. Various types of information about the game project, files, and processes are visible in this area.

Summary

In this chapter, we explored Amazon's new game engine, Lumberyard. The benefits of using Lumberyard for creating multiplayer AAA games was realized and included the ability to integrate with AWS for cloud computing and storage as well as integrating with Twitch for user engagement. We downloaded and installed Lumberyard, and installed additional software, SDKs, and plugins. Our final act in this chapter was to walk through the Lumberyard Editor's user interface, at a high level.

In the next chapter, we'll start planning our game for Lumberyard. This will be the game that we will build throughout the remainder of this book. We'll also explore Lumberyard's graphics capabilities and requirements.

2
Planning Your Game in the Lumberyard

We started the previous chapter with a brief overview of Lumberyard and looked at what its capabilities are. We then contemplated what we could do with Lumberyard and how it fits into the current game engine landscape. In that chapter, we looked at system requirements and compatibility issues, and took a cursory look at how Lumberyard interacts with cloud-based storage and services. In this chapter, we will look at Lumberyard's beta release history and explore how it impacts your development efforts. We will also preview the games sampled throughout this book. Also in this chapter, we will look at the game functionality that we will create in subsequent chapters, look at game design for Lumberyard games, and explore how to plan the development process.

In this chapter, you will:

- Understand the significance of Lumberyard's beta status
- Become familiar with the sample game content used throughout this book
- Understand the importance of game design and game design documents
- Understand how the *UI Editor* and *UI Animation Editor* can be used to create stunning user interfaces
- Understand how gameplay can be created in Lumberyard
- Understand how to create immersive game environments in Lumberyard
- Be able to plan your own Lumberyard development process

Beta software

As you likely know, the Lumberyard game engine is, at the time of this book's publication, in beta. What does that mean? It means a couple of things that are worth exploring.

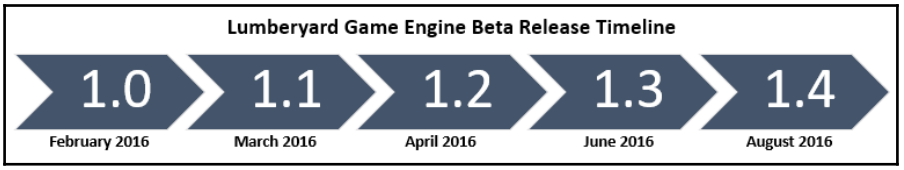
First, developers (that's you!) get early access to amazing software. Other than the cool fact of being able to experiment with a new game engine, it can accelerate game projects. There are several downsides to this as well. Here are the primary drawbacks of using beta software:

- Not all functions and features will be implemented. Depending on the engine's specific limitations, this can be a showstopper for your game project.
- Some functions and features might be partially implemented, not function correctly, or be unreliable. If the features that have these characteristics are not ones you plan to use, then this is not an issue for you. This, of course, can be a tremendous problem. For example, let's say that the engine's gravity system is buggy. That would make testing your game very difficult, as you would not be able to rely on the gravity system and not know if your code has issues or not.
- Things can change from release to release. Anything done in one beta version is apt to work just fine in subsequent beta releases. Things that tend to change between beta versions, other than bug fixes and improvements, are interface changes. This can slow a project up considerably, as development workflows you have adopted may no longer work.

In the next section, you will see what changes were ushered in with each sequential beta release.

Release notes

Amazon initially launched the Lumberyard game engine in February 2016. Since then, there have been several new versions. At the time of this book's publication, there were five releases: 1.0, 1.1, 1.2, 1.3, and 1.4. The following figure shows the timeline of the five releases:



Let's look at the major offerings of each beta release.

Beta 1.0

The initial beta of the Lumberyard game engine was released on February 9, 2016. This was an innovative offering from Amazon. Lumberyard was released as a triple-A cross-platform game engine at no cost to developers. Developers had full access to the game engine along with the underlying source code. This permits developers to release games without a revenue share and to even create their own game engines, although they're not distributable, using the Lumberyard source code as a base.

Beta 1.1

Beta 1.1 was released just a few short weeks after Beta 1.0. According to Amazon, there were 208 feature upgrades, bug fixes, and improvements with this release. Here are the highlights:

- Autoscaling features
- Component Entity System
- FBX Importer
- New Modular Gems
- Cloud Canvas Resource Manager
- New *Twitch ChatPlay* Features
- Initial support for Android and iOS

You will gain exposure to the **Component Entity System** throughout this book and learn about the FBX Importer in `Chapter 4`, *Creating 3D Characters*; Game Gems in `Chapter 8`, *Bringing Your Game to Life with Audio and Sound Effects* and `Chapter 9`, *Employing Cloud Computing and Storage*; **Cloud Canvas Resource Manager** in `Chapter 9`, *Employing Cloud Computing and Storage*; and **Twitch ChatPlay** in `Chapter 10`, *Engaging With Users Using Twitch*.

Beta 1.2

In just a few short weeks after beta 1.1 was released, beta 1.2 was made available. The rapid release of sequential beta versions is indicative of tremendous development work by the Lumberyard team. This also gives some indication as to the amount of support the game engine is likely to have once it is no longer in beta.

With this beta, Amazon announced 218 enhancements and bug fixes to nearly two dozen core Lumberyard components. Here are the main Lumberyard game engine components that were upgraded in beta 1.2:

- Particle Editor
- Mannequin
- Geppetto
- FBX Importer
- Multiplayer Gem
- Cloud Canvas Resource Manager

We will explore Mannequin in `Chapter 5`, *Animating Your Characters*; Geppetto and the FBX Importer in `Chapter 4`, *Creating 3D Characters*; the Multiplayer Gem in *`Chapter 7`*, *Creating Multiplayer Gameplay*; and the Cloud Canvas Resource Manager in `Chapter 9`, *Employing Cloud Computing and Storage*.

Beta 1.3

The three previous beta versions were released in subsequent months. This was an impressive pace, but likely not sustainable due to the tremendous complexities of game engine modifications and the fact that the Lumberyard game engine continues to mature. Released in June 2016, the Beta 1.3 release of Lumberyard introduced support for **Virtual Reality** (**VR**) and **High Dynamic Range** (**HDR**). We will talk about VR and Lumberyard in *`Chapter 12`*, *Stretching Your Lumberyard Wings*.

Adding support for VR and HDR is enough reason to release a new beta version. Impressively, this release also contained over 130 enhancements and bug fixes to the game engine. Here is a partial list of game engine components that were updated in this release:

- Volumetric Fog
- Motion Blur
- Height Mapped Ambient Occlusion
- Depth of Field
- Emittance
- Integrated Graphics Profiler
- FBX Importer
- UI Editor
- Flow Graph Nodes
- Cloud Canvas Resource Manager

Beta 1.4

At the time of this book's publication, the current beta version of Lumberyard was 1.4, which was released in August 2016. This release contained over 230 enhancements and bug fixes as well as some new features.

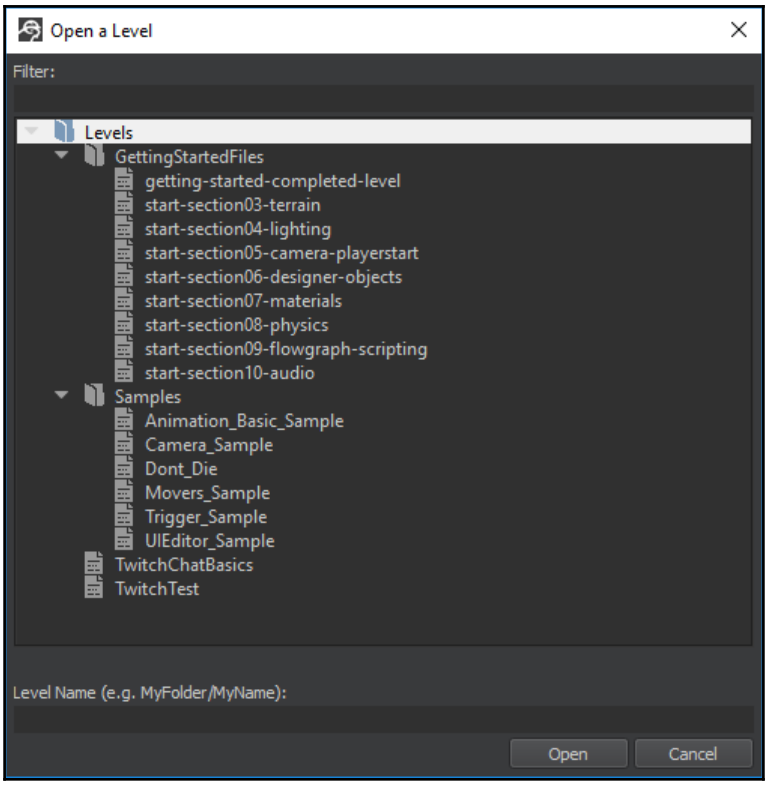
The primary focus of this release seemed to be on multiplayer games and making them more efficient. One example of increased efficiency is the addition of an in-editor Lua script editor. The result of the changes provided in this release is greater cost-efficiency for multiplayer games when using Amazon *GameLift*. You will learn about Amazon GameLift in `Chapter 7`, *Creating Multiplayer Game Play*.

Overview of sample content

Creating triple-A games is a complex process that typically involves a large number of people in a variety of roles, including programmers, designers, artists, and more. There is no industry average for how long it takes to create a triple-A game because there are too many variables, including budget, team size, game specifications, and individual and team experience. This being said, it is likely to take up to 2 years or more to create a triple-A game from scratch.

Triple-A, or AAA, games typically have very large budgets, large design and development teams, and large advertising efforts, and are largely successful. In a nutshell, triple-A games are large!

Two years is a long time, so we have shortened things for you in this book by using available game samples that come bundled with the game engine. As illustrated in the following screenshot, Lumberyard comes with a great set of starter content and sample games:



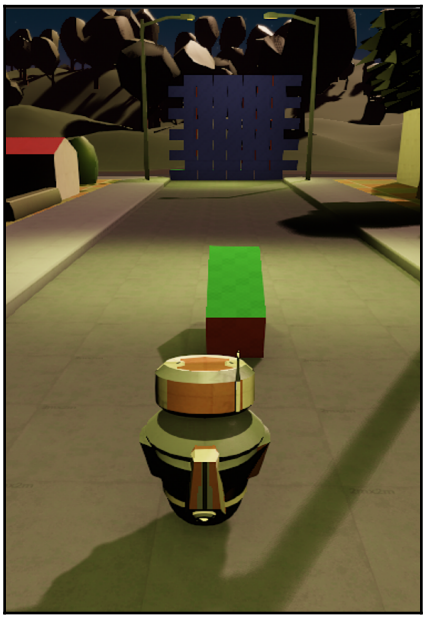
Starter content

When you first launch Lumberyard, you are able to create a new level or open an existing level. In the **Open Level** dialog window, you will find nine levels listed under **Levels** | **GettingStartedFiles**. Each of these levels presents a unique opportunity to explore the game engine and learn how things are done. Let's look at each of these next.

getting-started-completed-level

As the level name suggests, this is a complete game level featuring a small game grid and a player-controlled robot character. The character is moved with the standard *WASD* keyboard keys, is rotated with the mouse, and uses the Spacebar to jump.

The level does a great job of demonstrating physics. As is indicated in the following screenshot, the level contains a wall of blocks that have natural physics applied. The robot can run into the wall and fallen blocks and can use the ramp to launch into the wall. More than just playing the game, you can examine how it was created:

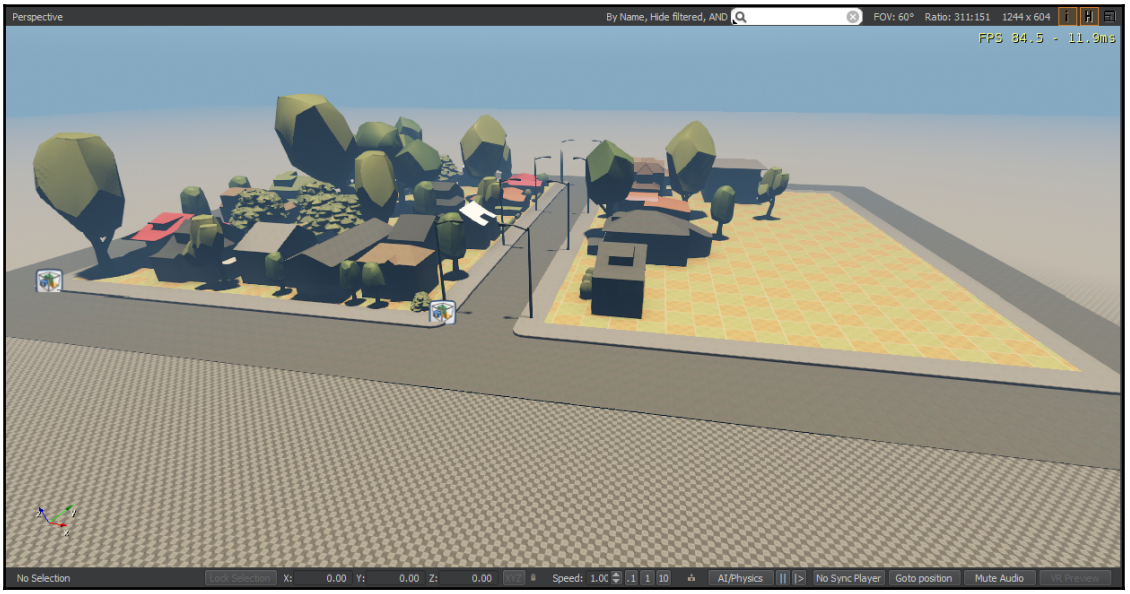


This level is fully playable. Simply use the *Ctrl* + *G* keyboard combination to enter **Game Mode**. When you are through playing the game, press the *Esc* key to exit.

This completed level can be further examined by loading the remaining levels, featured in this section. These subsequent levels make it easier to examine specific components of the level.

start-section03-terrain

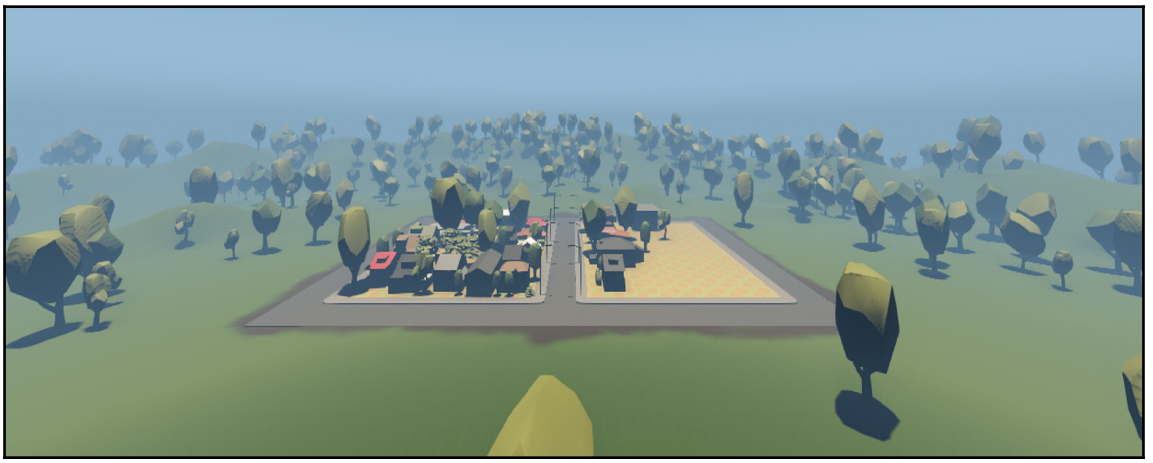
This section simply contains the terrain. As you can see in the following screenshot, the complete terrain is provided as well as additional space to explore and practice creating, duplicating, and placing objects:



This level is not playable and is provided to aid your learning of Lumberyard's terrain system.

start-section04-lighting

This level is presented with an expanded terrain to help you explore lighting options. There are environmental lighting effects as well as streetlamp objects that can be used to emit light and generate shadows in the game:



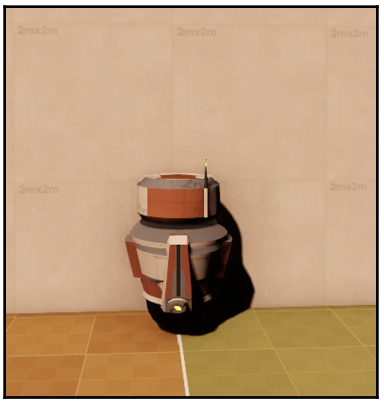
This level is not playable and is provided to aid your learning of Lumberyard's lighting system.

start-section05-camera-playerstart

This non-playable level is convenient for examining camera placement and discovering how it impacts the player's starting position on game launch. You will gain hands-on experience with cameras in `Chapter 3`, *Constructing an Immersive 3D Game World*.

start-section06-designer-objects

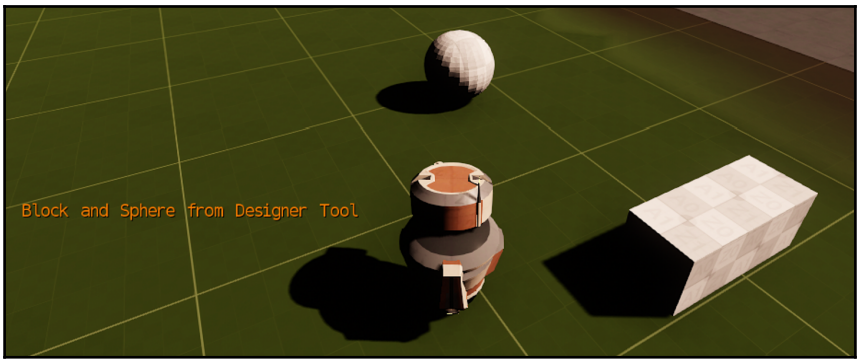
This level is playable, but only to the extent that you can control the robot character and explore the game's environment. You will notice that the graphic tiles are labeled with their size (for example, 2 m x 2 m):



With this level, you can focus on editing the objects.

start-section07-materials

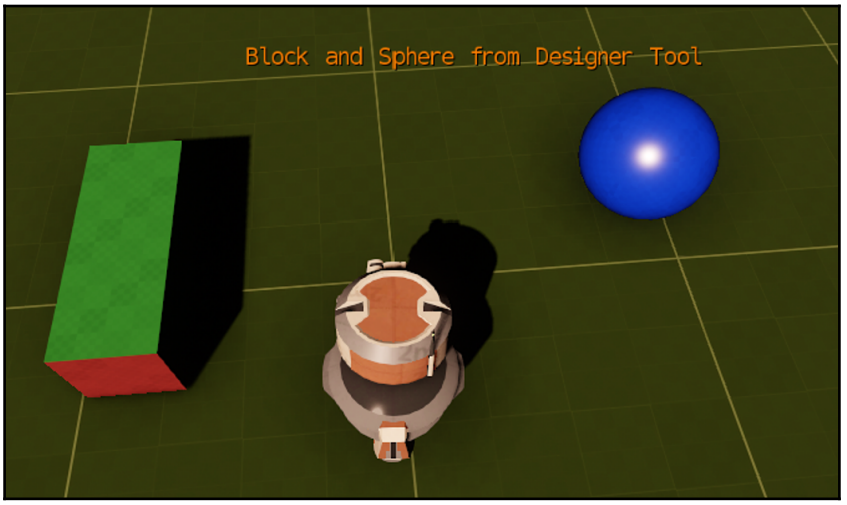
This level includes the full game environment along with two natively created objects: a block and a sphere. You can freely edit these objects and see how they look in **Game Mode**. This represents a great way to learn, as it is a no-risk situation. This simply means that you do not have to save your changes, as you are essentially working in a sandbox with no impact on a real game project:



This is a playable level that allows you to explore the game environment and preview any changes you make to the level.

start-section08-physics

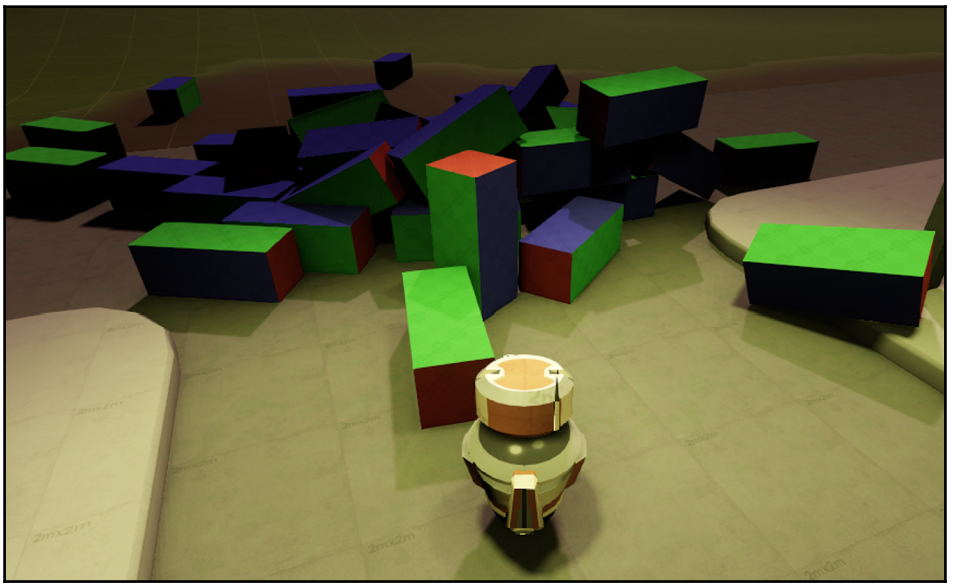
This starter level has the same two 3D objects (block and sphere) as the previous starter level. In this level, the objects have textures:



No physics have been applied to this level, so it is a good level to use to practice creating objects with physics. One option is to attempt to replicate the wall of stacked objects that is present in the completed level.

start-section09-flowgraph-scripting

This playable level contains the wall of 3D blocks that can, as illustrated by the following screenshot, be knocked over. The game's gameplay is instantiated with *Flow Graphs*, which can be viewed and edited using this starter level:



You will learn about *Flow Graphs* and the *Flow Graph Editor* in `Chapter 6`, *Creating Gameplay*.

start-section10-audio

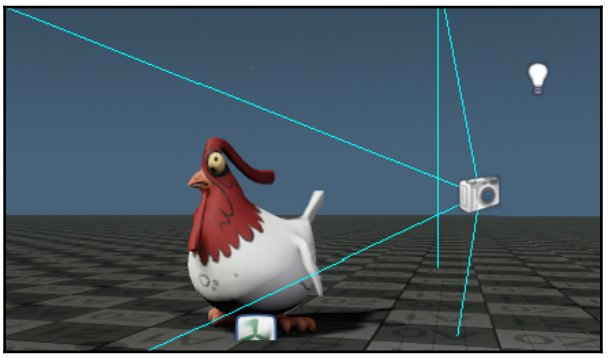
This final starter level contains the full playable game that serves as a testing ground for implementing audio in the game. You will learn about Lumberyard game audio in `Chapter 8`, *Bringing Your Game to Life with Audio and Sound Effects*.

Sample games

There are six sample unrelated game levels accessible using the **Open Level** dialog window; you will find nine levels listed under **Levels** | **Samples**. Each of these levels is a single-level game that demonstrates specific functionality and gameplay. Let's look at each of these next.

Animation_Basic_Sample

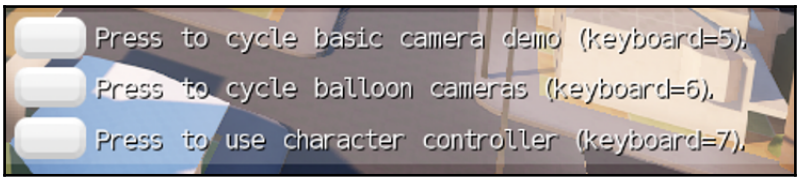
This game level contains an animated character in an empty game environment. There is a camera and light. You can play the game to watch the animated character's idle animation:



While this game level is not used in this book, we will make use of the animated character in `Chapter 4`, *Creating 3D Characters*, to gain hands-on experience with Geppetto, Lumberyard's animation tool.

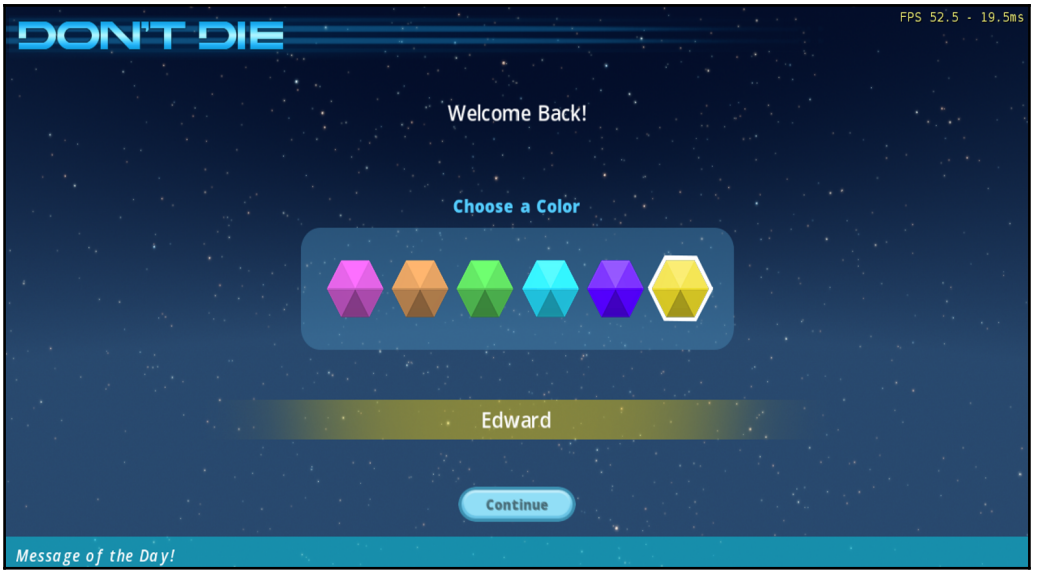
Camera_Sample

You will load this sample game in `Chapter 6`, *Creating Gameplay*, to help learn how to create gameplay. The sample game includes a **Heads Up Display** (**HUD**), which presents the player with three game modes, each with a different camera. Also in `Chapter 6`, *Creating Gameplay*, this game is used to further explore Flow Graphs and the Flow Graph Editor:



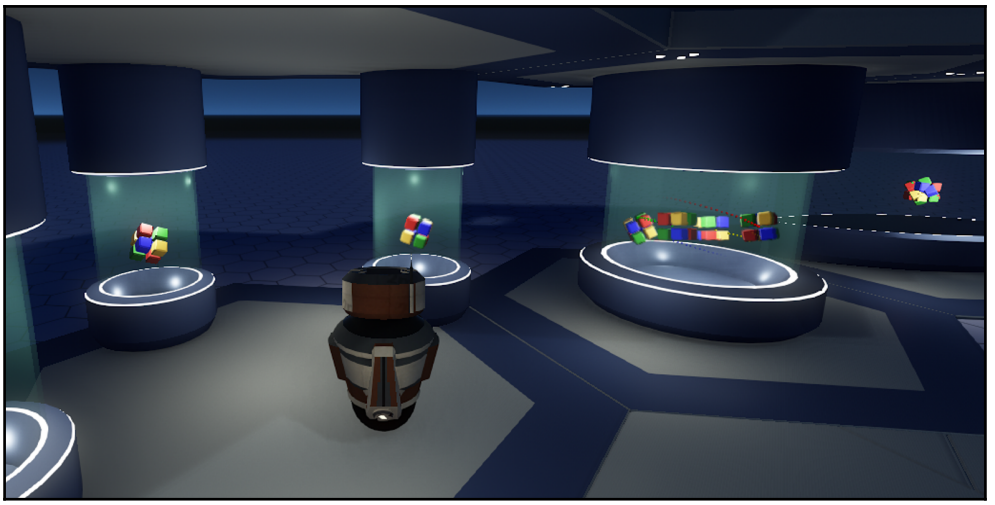
Dont_Die

The Don't Die game level provides a colorful example of full gameplay with an interactive menu system. The game starts with a **Press any key to start** message followed by a color selection menu depicted in the following screenshot. The selected color is applied to the spacecraft used in the game:



Movers_Sample

In `Chapter 5`, *Animating Your Characters*, you will explore this sample game to learn how to instantiate animations triggered by user input. We will also look at physics using this game sample. Using this game, you will also gain initial exposure to Flow Graphs and the Flow Graph Editor:



Trigger_Sample

This sample game provides several examples of **Proximity** and **Area Triggers**. As shown in the following screenshot, both identifying information and instructions are presented on the screen:

Here is a list of triggers instantiated in this sample game:

- Proximity trigger
  - Player only
  - Any entity
  - One entity at a time
  - Only selected entity
- Three entities required to trigger
- Area trigger
  - Two volumes use same trigger
  - Stand in all three volumes at the same time
  - Step inside each trigger in any order
  - Step inside each trigger in correct sequence

UIEditor_Sample

This sample game is not playable but provides a commercial-quality **Creating Gameplay** (**UI**) example. If you run the level in **Game Mode**, you will not have a game to play, but the stunning visuals of the UI give you a glimpse of what is possible with the Lumberyard game engine:

Game design and game design documents

Developing games, especially AAA games, takes a Herculean effort from a game project team. Sophisticated game engines such as Lumberyard go a long way in helping development teams with tools and efficiencies. Perhaps the single most important tool a game developer can have is the game design. More specifically, a **Game Design Document** (**GDD**). These documents contain the game's specifications, including a game's description that details the overall game concept, the game's genre or genres, a storyline synopsis, the target audience, and distribution platforms. GDDs also document the game's interface design, game mechanics, and immersion plan.

Not having a game design to guide the development process is like starting to build a house without knowing whether it is supposed to be constructed out of wood or brick, how many floors it should have, the square footage, and the style. That would be a nightmare house construction project. To avoid nightmare game development projects, we rely on GDDs. Let's examine each of the main components of the GDD and see how they relate to the Lumberyard game engine.

Game description

The game description typically contains an overview of the game, a synopsis of the storyline, game genre, target audience, and distribution platforms. We will review game genres and distribution platforms in Lumberyard in the following sections.

Game genre

In today's game development landscape, there are an increasing number of game genres. This is in part due to genre blending and the innovativeness and creativeness of game studios. Some of the most common game genres for AAA games are strategy/real-time strategy, shooters, action, adventure, sports, and simulation.

Lumberyard can support the development of any game genre. Since a genre is made up of gameplay and functionality, you are only limited by your team's imagination and ability to execute your plan.

Distribution platforms

AAA games are typically distributed to consoles and desktop/laptop computers. You can create games using Lumberyard that can be deployed on Xbox One and PlayStation 4 game consoles as well as computers running the Windows operating system.

There is currently limited support for mobile devices. As Lumberyard continues to mature, mobile support will increase. Support for computers running Mac and Linux operating systems is something we will likely see in the not-too-distant future.

You will learn more about distribution platforms in `Chapter 11`, *Providing Your Game to the World*.

User interface

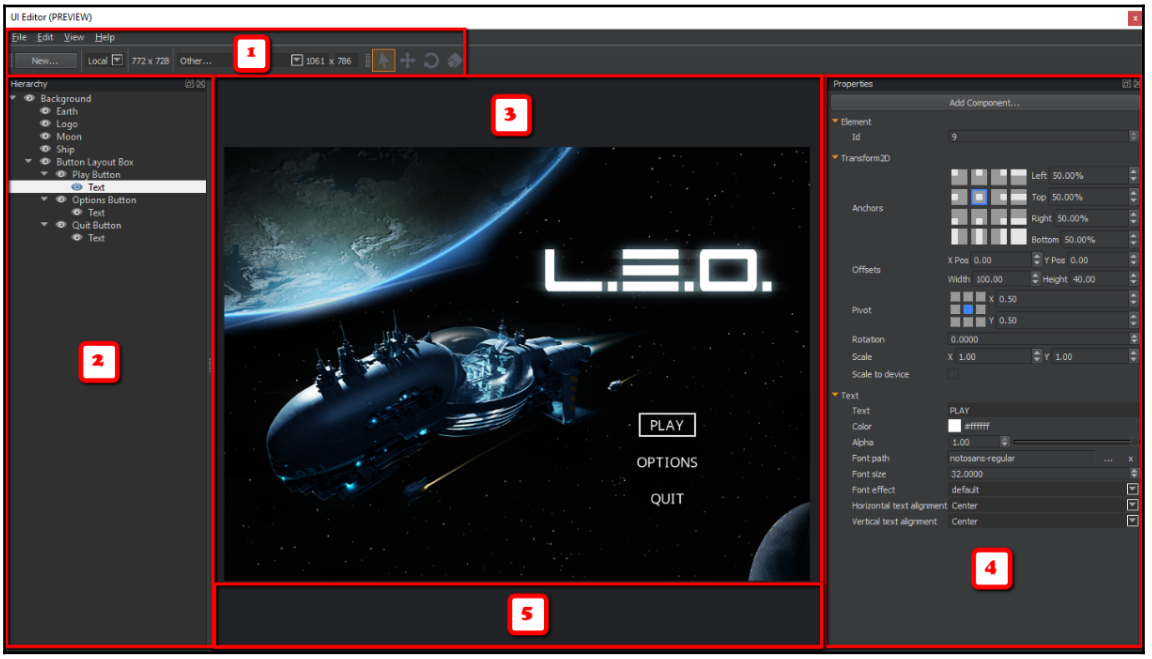
UIs for your game are usually mocked up in a GDD. This typically includes any screens that users will interact with. Depending on your game, this might only consist of a start menu or include additional interfaces such as inventory, game lobbies, dialog interfaces, and more.

Earlier in this chapter, you saw an interactive user interface with the **Don't Die** sample game. The **UIEditor Sample** game sample illustrated how professional a user interface can look in Lumberyard.

User input can be obtained from the user's keyboard, or a pointing device such as a mouse, trackball, or joystick. Game controllers are used to take user input on game consoles. For computers running the Windows operating system, an Xbox controller can also be used. When designing user interfaces in Lumberyard, it is important to determine what forms of input your game will accept from users. Throughout this book, you will gain hands-on experience in defining user input from a computer keyboard.

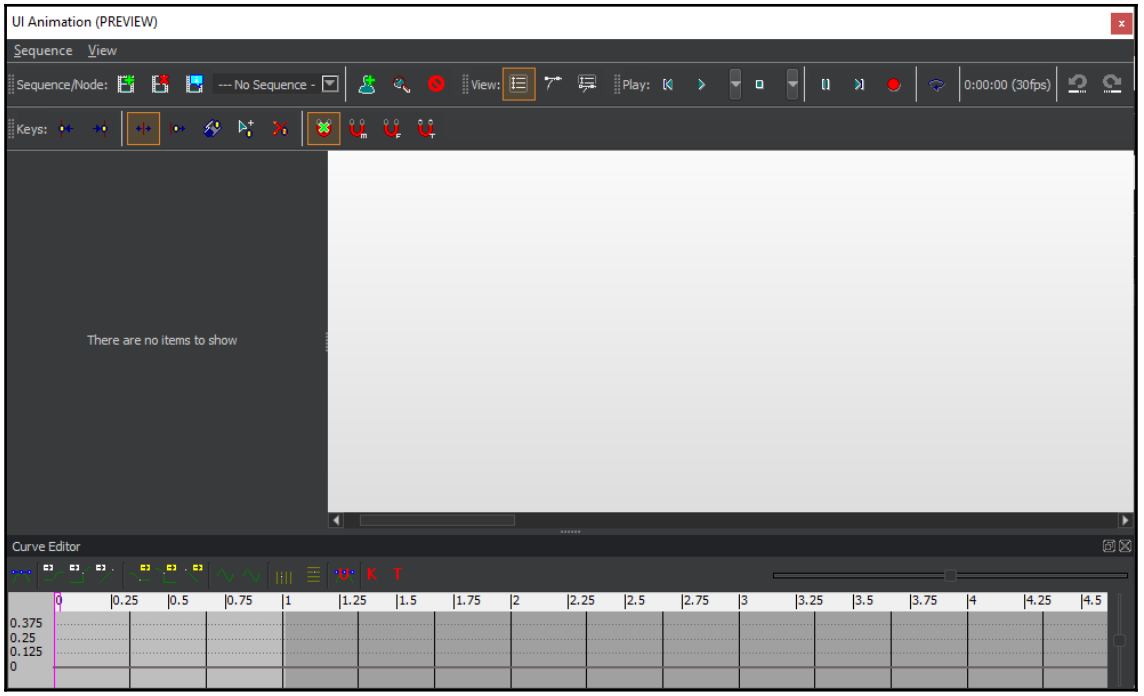
Lumberyard's UI Editor

Lumberyard has a **UI Editor** that streamlines the creation of user interfaces. The **UI Editor** is accessible via the **View** | **Open View Pane** | **UI Editor** menu selection. As illustrated in the following screenshot, the **UI Editor** has five panels:



- **1**– Menus and toolbar.
- **2** – **Hierarchy** pane, this pane contains a hierarchical list of your user interface components. You can arrange these items to make it easier to locate components and increase your efficiency when editing the interface.
- **3** – UI viewport. This is where you can preview your interface and select and move your objects.
- **4** – Contextual properties pane. Selecting a UI component will populate this pane with the selected object's properties. The properties can be edited in this pane.

- **5** – UI **Animation Editor**. Depending on your version of Lumberyard, you might have access to the UI Animation Editor in the pane below the viewport. If that pane is not visible, select **View** | **Animation Editor** from the **UI Editor** menu. As shown in the following screenshot, the UI Animation Editor is fairly complex. It is presented here to show that animation sequences can be applied to UI components. Animating UI components is something that you can consider when you design and develop your game:

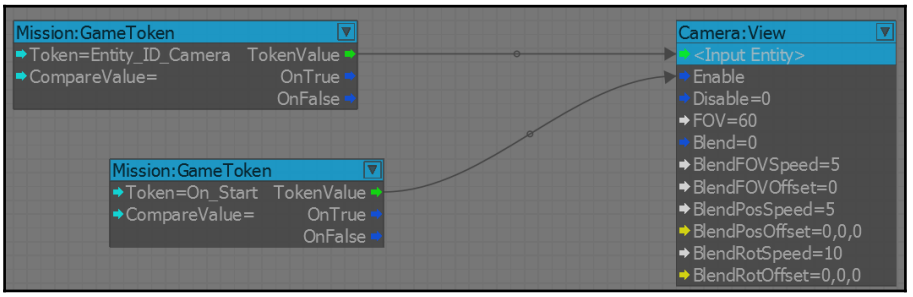


Creating a gameplay in Lumberyard

A gameplay is the manner in which players interact with other players and with your game. It is sometimes referred to as game mechanics and game functionality. Gameplays are different from user interfaces.

You might create a gameplay so that players can pick items up, put them down, and interact with them. Maybe players can shoot other players, non-player characters (AI players), and other objects. Jumping, running, climbing, crawling, and sliding are also examples of gameplay. Players can throw grenades, fix a truck, destroy a bridge, and dig a hole. The list is unlimited.

In Lumberyard, gameplay is created with Flow Graphs. Flow Graphs are a visual scripting system and are created using Lumberyard's Flow Graph Editor. The Flow Graph Editor is accessible through the **View** | **Open View Pane** | **Flow Graph** pull-down menu. A sample Flow Graph is provided in following screenshot to show how relatively easy they are to read:



You will learn more about Flow Graphs and gain hands-on experience with them in `Chapter 6`, *Creating Gameplay*.

Creating immersive games in Lumberyard

One of the reasons for the success of AAA games is that they create immersive game experiences for players. Immersive games are those that result in the player being so wrapped up in a game that they are no longer aware of the outside world. When a player is immersed in a game, they are *in* the game and time passes unnoticed. Players can have physical reactions to in-game events when they are immersed. Physical reactions include increased heart rate, tighter grips on a controller, facial expressions, and more.

You can create immersive games in Lumberyard in a couple of key ways. First, the game world should be consistent. This is more than just realism. Even cartoonish and futuristic games can be immersive. The goal is to ensure all game components are consistent and realistic for the game itself.

Natural user controls

Another way to help ensure the games you create with Lumberyard are immersive is to make the user controls natural so that they are not distracting. If a player is trying to figure out a long control key combination for a fight move, their focus is off the game world and on the controller. This should be a crucial consideration.

Game audio

Game audio is one of the most significant contributors to immersion. Sounds should be realistic and consistent for the game. Appropriate ambient sounds, background sounds, and sound effects should be used in the game. Achieving excellence with game audio is a specialized skill.

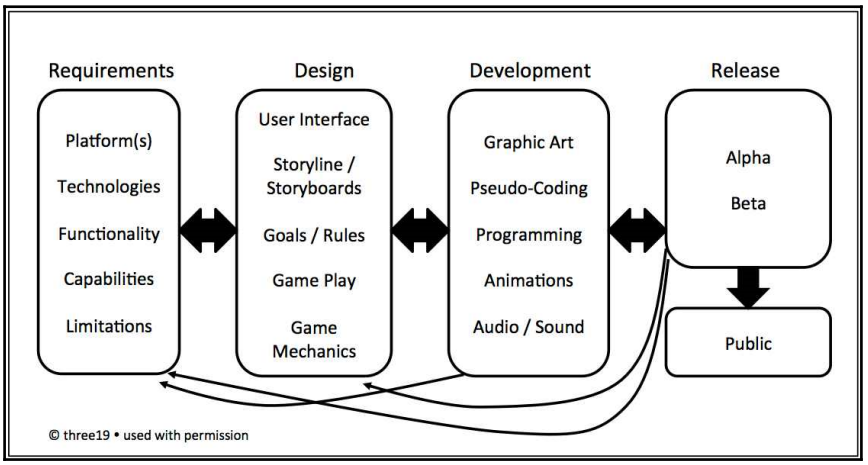
It is important to pay attention to the volume of individual sounds. Background music, for example, should not drown out sound effects or AI player spoken dialog. You can use fading and fall-off techniques to control the volume of individual audio clips.

Using surround sound is a great contributor to game immersion. If a truck is approaching the player's position from the left, then the sound of that truck should come from the left speaker and gravitate toward center and move off to the right, as appropriate for the truck's path. These finite considerations go a long way in helping to provide an immersive game environment for the player.

You will learn about Lumberyard's Audio System in `Chapter 8`, *Bringing Your Game to Life with Audio and Sound Effects*.

Planning your Lumberyard development process

Each game development process is unique. Even game studios that are releasing a sequel to a successful game will have differences in how they develop each game. Team composition, goals, timelines, and tools are all subject to change. The following game development process diagram provides an overview of how the major components and sub processes might work together:

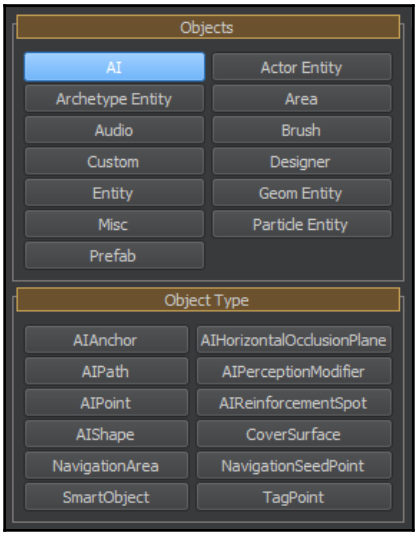


Here are some Lumberyard tools that can be used by individuals or teams on your game project to simultaneously work on major components of the game. These tools are presented in alphabetical order so no importance or significance levels should be assumed. In some cases, the tools are embedded in a project component description. So, not all items listed below are names of specific Lumberyard tools.

There is some overlap, as several different teams will need to use more than one tool. As appropriate, references to this book's chapters are provided so you know where to find additional information on each tool.

AI System

Lumberyard's **AI System**, or Artificial Intelligence System, allows you to create game entities that exhibit AI behaviors. As illustrated in the attached screenshot from the **RollupBar**, there are 12 AI object types in Lumberyard. You learned about the **RollupBar** in `Chapter 1`, *Welcome to the Lumberyard*, and will gain hands-on experience with that section of the Lumberyard game engine throughout the remaining chapters:



Selecting an AI object type reveals relevant properties. When working on the AI System for your game, the above interface is used in conjunction with Flow Graphs to create your game's AI behaviors.

Amazon Web Services

Amazon Web Services (**AWS**) is a family of cloud-based scalable services to support, in the context of Lumberyard, your game. AWS includes several technologies that can support your Lumberyard game. These services include:

- Cloud Canvas
- Cloud Computing
- GameLift

- **Simple Notification Service** (**SNS**)
- **Simple Query Service** (**SQS**)
- **Simple Storage Service** (**S3**)

You will learn more about AWS, including creating a personal account and gaining hands-on experience, in `Chapter 7`, *Creating Multiplayer Gameplay*, `Chapter 9`, *Employing Cloud Computing and Storage*, and `Chapter 12`, *Stretching Your Lumberyard Wings*.

Art asset creation

Game assets include graphic files such as materials, textures, color palettes, 2D objects, and 3D objects. These assets are used to bring a game to life. A terrain, for example, is nothing without grass and dirt textures applied to it. Much of this content is likely to be created with external tools.

One internal tool used to implement the externally created graphical assets is the material editor. Assets and the material editor are explored in `Chapter 3`, *Constructing an Immersive 3D Game World*.

Audio system

Lumberyard has an Audio System that controls how in-game audio is instantiated. No audio sounds are created directly in Lumberyard. Instead, they are created using Wwise Software by Audiokinetic. You will learn more about the Audio System and Wwise in `Chapter 8`, *Bringing Your Game to Life with Audio and Sound Effects*.

Because audio is created externally to Lumberyard, a game project's audio team will likely consist of content creators and developers that implement the content in the Lumberyard game.

Cinematics System

Lumberyard's **Cinematics System** can be used to create cut-scenes and promotional videos. With this system, you can also make your cinematics interactive. The Cinematics System is covered in `Chapter 12`, *Stretching Your Lumberyard Wings*.

Flow Graph System

Lumberyard's **Flow Graph System** is a visual scripting system for creating gameplay. This tool is likely to be used by many of your smaller teams. It can be beneficial to have someone who oversees all Flow Graphs to ensure compatibility and standardization.

The Flow Graph System is first explored in `Chapter 5`, *Animating Your Characters*, and hands-on experience with Flow Graphs is provided throughout the book.

Geppetto

Geppetto is Lumberyard's character tool. A character team will likely create the game's characters using external tools such as *Autodesk Maya* or *Autodesk 3DS Max*. Using those systems, they can export the necessary files to support importing the character assets into your Lumberyard game. Lumberyard has an `FBX Importer` tool that is used to import characters created in external programs.

You will learn more about Geppetto in `Chapter 4`, *Creating 3D Characters* and `Chapter 5`, *Animating Your Characters*.

Mannequin Editor

Animating objects, especially 3D objects, is a complex process that takes artistic talent and technical expertise. Some projects incorporate separate teams for object creation and animation. For example, you might have a small team that creates robot characters and another team that generates their animations.

The **Mannequin Editor** is discussed in `Chapter 4`, *Creating 3D Characters* and `Chapter 5`, *Animating Your Characters*.

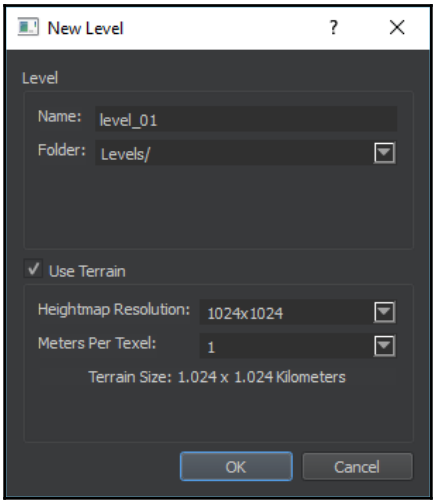
Production team

The production team is responsible for creating builds and distributing releases. They will also handle testing coordination. One of their primary tools will be the *Waf Build System*.

Distribution details are provided in `Chapter 11`, *Providing Your Game to the World*, and the Waf Build System is examined in `Chapter 12`, *Stretching Your Lumberyard Wings*.

Terrain Editor

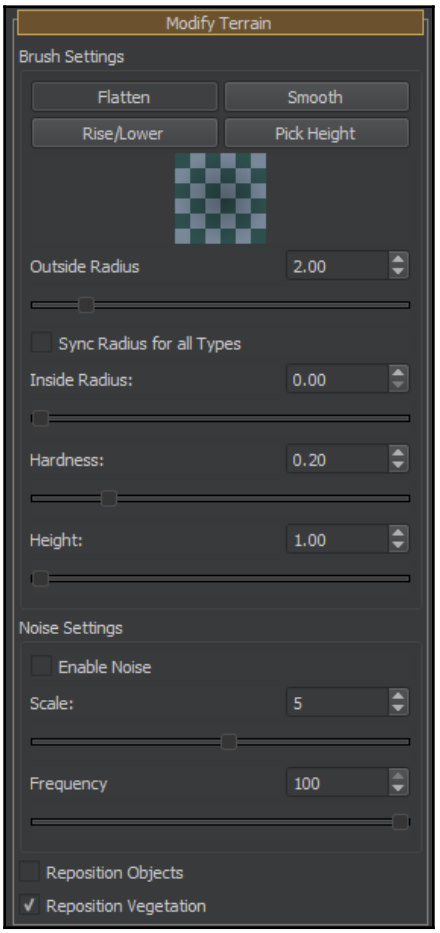
A game's environment consists of terrain and objects. The terrain is the foundation for the entire game experience and is the focus of exacting efforts. The creation of a terrain starts when a new level is created. As you can see in the following screenshot, the **HeightMap Resolution** is the first decision a level editor, or the person responsible for creating terrain, is faced with:



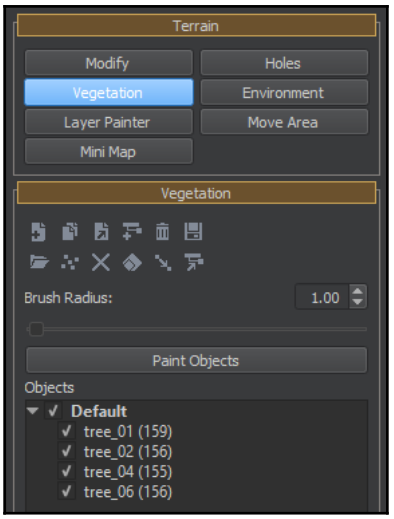
When **Terrain** is selected from the RollupBar, there are seven categories associated with the **Terrain**:

Each of these buttons gives you access to finite aspects of your terrain. Here is a brief overview of the functionality available via each of these buttons:

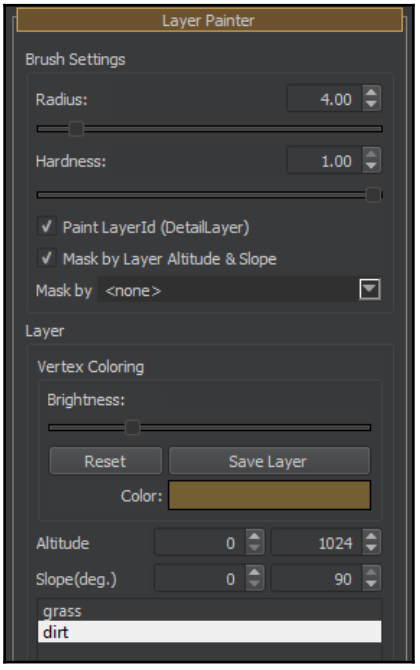
- **Modify** – Using custom brushes, you can flatten, smooth, raise, and lower your terrain:



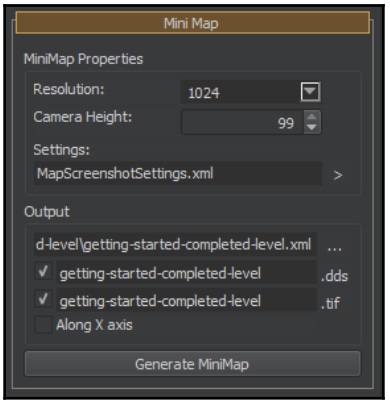
- **Vegetation** – Vegetation objects can be painted on your terrain with several brush properties. As illustrated in the following screenshot, there are several tool icons associated with **Vegetation**:



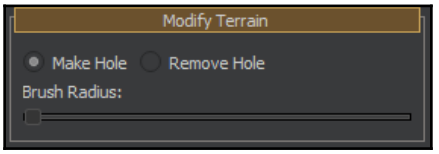
- **Layer Painter** – Using brushes, colors and materials can be painted onto your terrain:



- **Mini Map** – Creating mini maps is usually a complex ordeal in game engines. Lumberyard makes this relatively easy. As you can see from the following screenshot, you can edit a few parameters and click the **Generate MiniMap** button to get started:



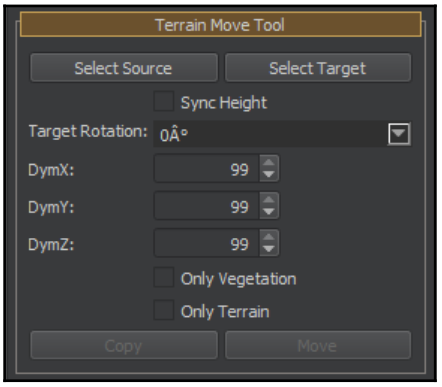
- **Holes** – The **Holes** tool is the most simplistic of the **Terrain** tools. You have the option of making or removing a hole and you can adjust the radius of your brush. From there, you simply paint holes on or off of your terrain:



- **Environment** – The **Environment** section of **Terrain** represents the most robust set of parameters to tailor your terrain. These parameters are grouped into the following 12 sections:
  - **Fog**
  - **Terrain**
  - **EnvState**
  - **VolFogShadows**
  - **CloudShadows**
  - **ParticleLighting**
  - **SkyBox**
  - **Ocean**
  - **OceanAnimation**

- **Moon**
- **DynTexSource**
- **TotalIllumination v2**

- **Move Area** – This function gives you the opportunity to move or copy sections of terrain. You can also select whether you are performing this operation for the terrain only, only the vegetation, or both:



Terrain creation and editing are covered extensively in `Chapter 3`, *Constructing an Immersive 3D Game World*.

Twitch ChatPlay system

Twitch integration represents exciting game possibilities. As you will learn in `Chapter 10`, *Engaging With Users Using Twitch*, Twitch integration allows you to engage your game's users in unique ways. In `Chapter 10`, *Engaging With Users Using Twitch*, you will create a game from scratch that allows Twitch users to type commands, in a chat window, which impact in-game actions.

UI Editor

Creating a user interface is often, at least on very large projects, the responsibility of a specialized team. This team, or individual, will create the user interface components on each game level to ensure consistency. Artwork required for the user interfaces is likely to be produced by the asset team.

Summary

In this chapter, we looked at Lumberyard's beta release history and explored how specific components of the Lumberyard game engine have evolved. We also looked at how the beta release schema could impact your development efforts. In addition, we previewed the games sampled throughout this book. Also in this chapter, we discussed the importance of game design and game design documents, previewed the **UI Editor** and **UI Animation Editor** for creating stunning interfaces, and explored how gameplay can be created in Lumberyard. We concluded the chapter with a look at how to create immersive game environments in Lumberyard, and how to plan your own Lumberyard development process.

In the next chapter, we will thoroughly explore terrains and gain hands-on experience, creating and modifying game level terrain. We will also look at how levels are used in Lumberyard, how to create terrain texture layers, and how to assign material to texture layers. You will follow step-by-step procedures to add vegetation to your game world, add water, create special environmental effects, add a camera, and add a player controller to your game.

3
Constructing an Immersive 3D Game World

In the previous chapter, we defined our game's design. Specifically, we documented the game world we want to create, the player-characters, the non-player-characters, the game objects, gameplay, and our immersion plan. Throughout the remainder of this book, we will make those plans come to life. We'll start with the creation of our game world. That is the focus of this chapter.

Our game world, also known as the game environment, will have several key features to include, such as trees, rivers, hills, mountains, light sources, and shadows. And, in order to test our game, we will create a player-character and a camera.

The concept of immersion is important to grasp. An immersion plan is typically part of a game's design documentation. This might not be a formal part of a document, but it is an overarching consideration. We want players to immerse themselves in our game, also referred to as *losing themselves in the game*. To achieve this level of immersion, we must create believable and consistent audio and visuals. We also need a way for players to navigate our game world, to control the player-character, and easily interact with the game objects.

In this chapter, you will:

- Explore how levels are used in Lumberyard
- Be able to create a blank level
- Be able to create terrain texture layers
- Know how to assign materials to texture layers
- Be able to add vegetation to your game world
- Be able to add a water feature to your game world

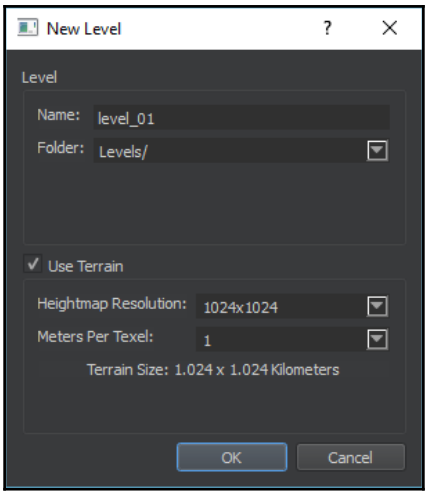
- Know how to create special environmental effects
- Be able to add a camera
- Be able to add a player controller
- Know how to test a level in game mode

Your first level

Most games are made up of several levels. Using levels in our games gives us the opportunity to nicely organize our game assets. How levels are implemented depends on the individual game. Levels might be progressive in nature with increasing difficulty. Levels might also simply contain different areas of the overall game world. In some games, the player can travel between levels at will, while others require specific in-game achievements before additional levels are unlocked.

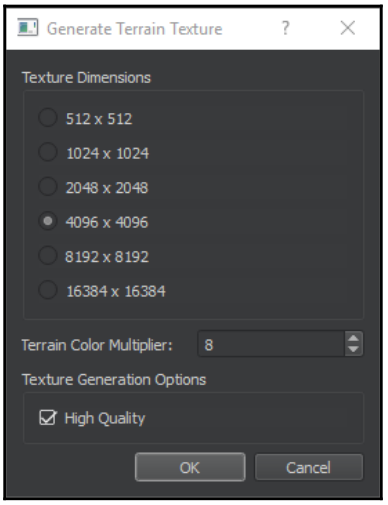
Lumberyard uses levels to organize game projects. Using the following steps, we will create a new game level. Once our level is created, we'll modify and configure it for our game:

1. Launch the **Lumberyard Editor**. Remember, Lumberyard is a large program and could take longer to load than you are accustomed to with other software packages.
2. When the **Welcome to Lumberyard Editor** dialog window appears, select the **New level** button. On subsequent launches of the editor, you will simply select your level to open.
3. In the **New Level** dialog window, rename the level **level_01**, ensure the **Use Terrain** option is checked, and accept the defaults for **Heightmap Resolution** and **Meters Per Texel** by clicking the **OK** button:



A `heightmap` is a file that contains the surface geometry for a given area. You can create your own `heightmap` as illustrated in this chapter or obtain your heightmap externally and import it to your Lumberyard level.

4. Next, you will be presented with the **Generate Terrain Texture** dialog window. Click the **OK** button to accept the default settings:



Our level has been created. Currently, it is a `4,096 x 4,096` blank area. As you will see, these dimensions make for a very expansive game environment. The dimensions of your game world are significant. Going with the maximum setting is not always the best idea. If you are making an arena combat game, you might consider using a much smaller size, such as `512 x 512`. There are other considerations as well. Larger level dimensions usually mean there is more content to build. Also, if you are making a **Massive Multiplayer Online Role-Playing Game** (**MMORPG**), you will want to ensure your networking solution supports the size of the level. You can perform testing to see what will work best for your game.

Throughout the rest of this chapter, we'll turn our level into a game world.

Creating terrain

Terrain is the surface area of our game world. It will consist of the ground, mountains, water, and other earthly features. Careful planning is recommended when determining how the terrain will look. Your game's players will be fully exposed to what you create and will want to be impressed. If you are making a combat game set in the modern world, having purple grass and trees with feathers might not make sense. Our goal with our terrain is to have it support immersion and not be a distraction.

We'll start by creating terrain texture layers and then assigning materials to those layers.

Creating terrain texture layers

Follow these steps to first create, and then create textures for, the terrain:

1. Using the pull-down menus, select **View** | **Open View Pane** | **Material Editor**.
2. Leave the **Material Editor** window open and select **View** | **Open View Pane** | **Terrain Texture Layers**.
3. In the **Terrain Texture Layers** window, click the **Add Layer** option three times. We will create three different textures.
4. Change the layer names from **NewLayer** to `rocks`, `soil`, and `grass`. You can change the names by double-clicking them.

Assigning materials to texture layers

Now that we have created our texture layers, we can assign materials to them. Follow these steps to perform that task:

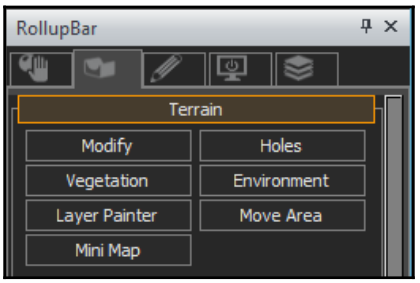
1. Click on the **Material Editor** window/tab.
2. In the left panel of the **Material Editor** window, expand **Materials**.
3. Expand the **gettingstartedmaterials** folder.
4. Select the **ground_stones_rough** material.
5. Navigate back to the **Terrain Texture Layers** window/tab.
6. In the **Terrain Texture Layers** window, select the **rocks** layer.
7. In the **Layer Tasks** section of the left pane, click **Assign Material**. This assigns the selected material to the rocks terrain texture layer. We will perform the same operation for the remaining two layers.
8. In the **Material Editor** window, select the **gs_ground_01** material.
9. In the **Terrain Texture Layers** window, select the `soil` layer.
10. Still in the **Terrain Texture Layers** window, select the **Assign Material** link in the **Layer Tasks** section in the left pane.
11. In the **Material Editor** window, select the **gs_grass_01** material.
12. In the **Terrain Texture Layers** window, select the `grass` layer.
13. Still in the **Terrain Texture Layers** window, select the **Assign Material** link in the **Layer Tasks** section in the left pane.

It is a good idea to save your project frequently. You can do this by selecting **File** | **Save** from the pull-down menu as well as by using the *Ctrl* + *S* keyboard combination.

Painting the terrain

Next we will apply the materials to the terrain. We accomplish this by painting with brushes. First, close the **Material Editor** and **Terrain Texture Layers** windows. Closing these windows unclutters our work environment. If you have multiple monitors, you can leave them open and out of the way. You can also open and close them whenever you choose.

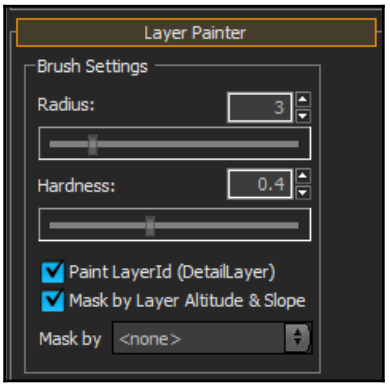
We will be working with the **RollupBar**, which is, by default, located in the right-hand section of the Lumberyard interface. The second tab on that bar is the **Terrain** tab. Select that tab:



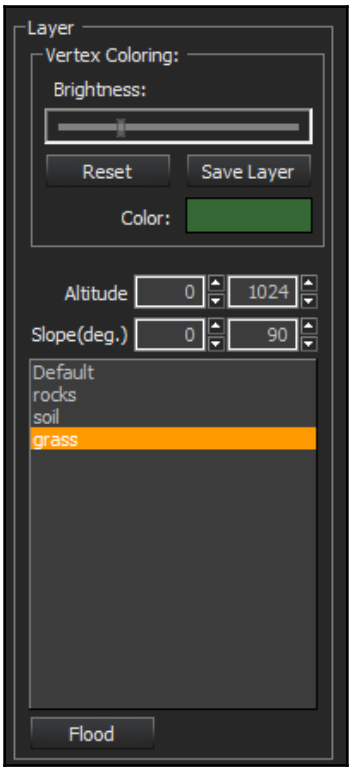
Lumberyard provides us with a great set of **Terrain** tools. We will look at each tool when we first use them. For now, we are going to work with the **Layer Painter** feature. Click that button to open the **Layer Painter** subpanel in the **RollupBar** panel.

The top portion of the **Layer Painter** presents us with settings for both **Brush** and **Layer**. The brush settings provide us with options for **Radius** and **Hardness**. The radius refers to the size of the brush from a value of `1` to `1,000`. The larger the **Radius**, the larger the area that you will paint at one time. The brush **Hardness** is analogous to how firmly you would press a real brush down on the surface you are painting. **A Hardness** value of `1` means the brush is being pressed all the way down. The smaller the number, the lighter the brush strokes will be. This is something you can experiment with to determine what value, from 0 to 1, you prefer. In order to perform experiments where you can clearly see the difference, you can increase the zoom level using your mouse scroll wheel in the Viewport.

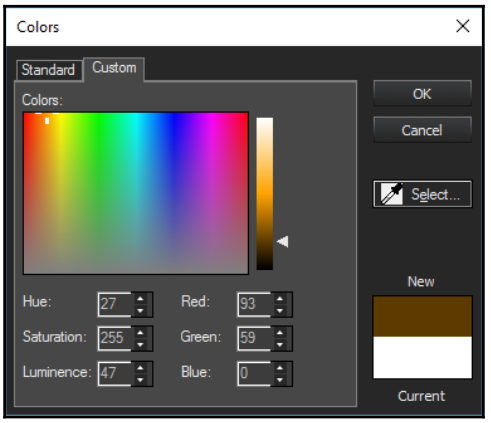
For your first use of the **Layer Painter**, set the **Radius** to `3` and the **Hardness** to `0.4`. Leave the remaining settings in this subpanel at their default values:



The bottom portion of the **Layer Painter** features is **Layer**. There are four areas of this interface: **Vertex Coloring**, **Altitude** and **Slope(deg.)**, a list of layer textures, and a **Flood** button. We will cover our terrain with soil material. But first, we must select a color for the dirt:



Click the **soil** texture in the list. Then, click the white color box. That will bring up the **Colors** dialog window. Select a shade of brown for the soil. You can enter *RGB* values of `93`, `59`, and 0 directly or select from the color interface to identify the color you desire. Once you have selected a soil color, click the **OK** button:



Now we can simply click the **Flood** button to completely fill our game world with our soil. Hence, we click the **Flood** button next.

Do not be alarmed if your game world does not look like you envisioned it yet. We are just getting started and have a lot to do before we have a polished game world.

At this point, you should feel confident in using the **Layer Painter**. As you go through the rest of this chapter, you should experiment with the functionality presented. You will be provided with access to an end-of-chapter Lumberyard project file to start the next chapter.

Configuring the game world

Now that our terrain textures are set, we can begin to make our game world look more appealing. We'll accomplish this by adding colored textures to our terrain, adding vegetation throughout the world, sculpting segments of the terrain to create hills and mountains, and adding a water feature.

Lumberyard offers a lot of flexibility in creating game worlds. We can create our own blank `heightmap` as we did in the previous section, import a `heightmap`, apply textures, and use virtual brushes to paint our world.

Adding color

In this section, we will look at two ways to add color to our world. First, we will paint colors onto our terrain with virtual brushes. Then, we will add vegetation to add additional realism and color.

You can and should experiment with the functionality described in this section until you feel confident with it. The completed level file will be available to download on the book's website.

Painting our terrain

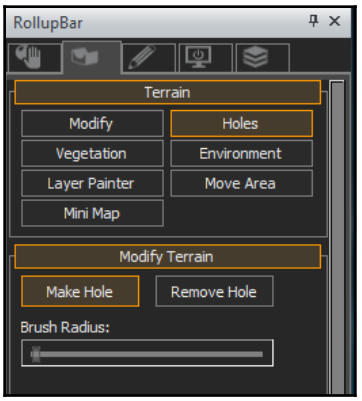
The steps required to paint textures onto a terrain are relatively straightforward. For this function, you will use the **RollupBar** interface. Specifically, you will use the **Layer Painter** feature set of the **Terrain** tab. Here you can modify the brush size by adjusting the **Radius**. We also can select a **Hardness** value to define how firm we want the edges. Typically, these settings are fine-tuned throughout the terrain creation process and are not static for the entire terrain.

Once the **Radius** and **Hardness** settings are to your liking, you will select a layer to use as your paint. We previously created the `rock`, `soil`, and `grass` layers. Select one of them, and then turn your attention to the Viewport. The mouse is now your paintbrush. Turn your focus to the Viewport. To start painting, simply left-click with your mouse and drag the mouse cursor around to paint the area you desire.

You can easily navigate in the Viewport window, also referred to as the perspective, with the following controls:

- Mouse scroll wheel – Zoom in and out
- Right-click – Pan around current location
- *Alt* + right mouse button – Zoom in and out

You can create holes in your terrain by selecting **RollupBar** | **Terrain Tab** | **Holes**. As you can see in the following screenshot, you can reverse any holes created by using the **Remove Hole** button. This feature works the same as painting. You adjust the **Brush Radius** with the slider interface and then click and paint the holes onto your environment in the Viewport:

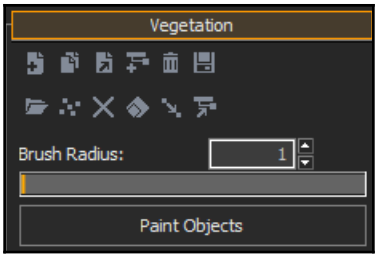


Adding vegetation

Another method of adding color to a game environment is to add vegetation. Vegetation comes in many forms, shapes, and sizes. Lumberyard comes with some default vegetation objects such as trees, shrubs, and brush. You can create your own vegetation in external graphics programs such as Blender, Maya, and 3DS. You can also obtain 3D models of vegetation from online digital marketplaces.

Let's get started with vegetation. First, let's take a look at Lumberyard's interface for vegetation. You can access this interface in the **RollupBar** by selecting **Terrain** | **Vegetation**.

In the top part of the interface, shown in the following screenshot, there are 12 icons that give you access to tools required to create, edit, and manage vegetation objects:



Here is an overview of the 12 vegetation icons along with their associated functionality:

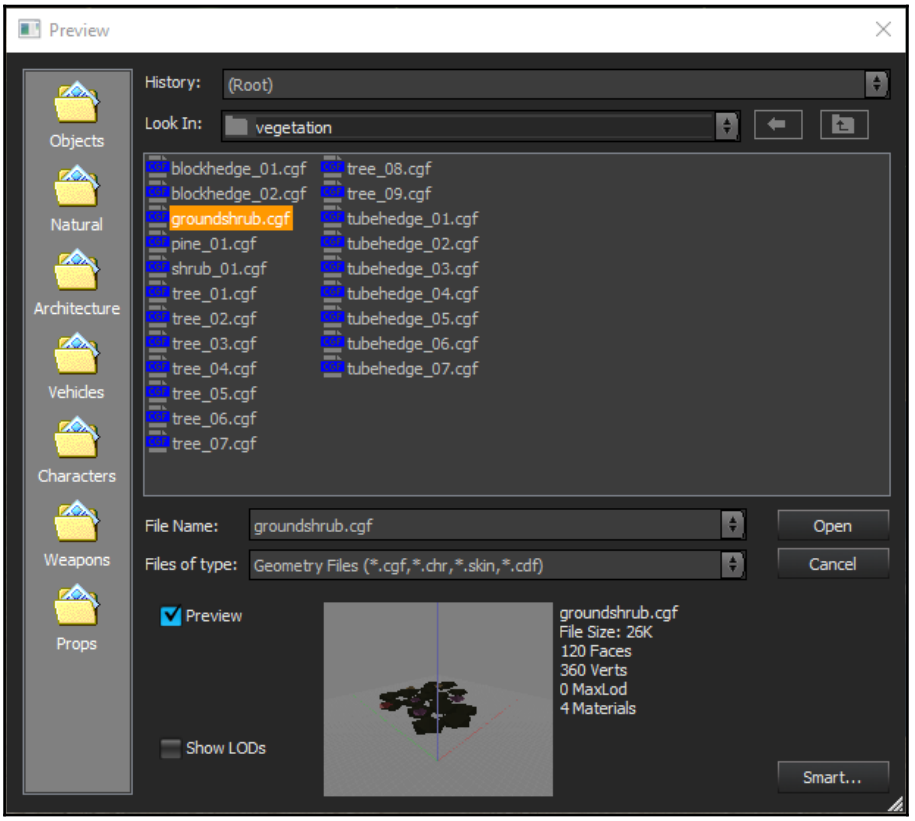
Icon	Functionality
	Add vegetation object
	Clone vegetation object
	Replace vegetation object
	Add vegetation category
	Remove vegetation object
	Export vegetation
	Import vegetation
	Distribute vegetation on whole terrain
	Clear vegetation
	Scale Vegetation
	Merge vegetation
	Put selection to category

Follow these steps to add vegetation to your game world:

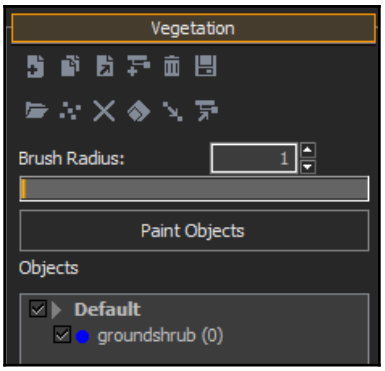
1. Select **RollupBar** | **Terrain** | **Vegetation** | **Add Vegetation Object**. This will bring up a **Preview** window.

2. In the **Preview** window, navigate to the following path: **objects** | **styletown** | **natural** | **vegetation**. This displays several vegetation objects to include, such as trees, hedges, and shrubs.

   Selecting an object in the **Preview** window provides descriptive information, including a thumbnail image. See the example in the following screenshot, where a ground shrub is selected. In addition to the preview thumbnail image, the following information is provided: file size, number of faces, number of vertices, maximum **Level of Detail** (**LOD**), and the number of materials:



3. Select the **groundshrub** object by clicking on its filename and click the **Open** button. This closes the **Preview** window and adds the ground shrub to the **Vegetation** object list:



When you select an object in the **Objects** list, you will see a host of settings that you can change. These settings are listed in a pane directly below the **Objects** pane.

4. Ensure you have the **groundshrub** object selected. This tells Lumberyard the object you want to paint.
5. Click the **Paint Objects** button. This makes the vegetation object the painting object.
6. In the Viewport, paint the vegetation object on your terrain.

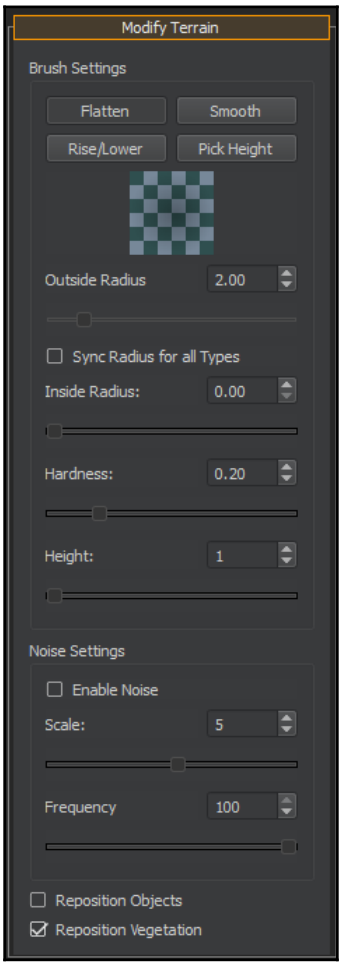
You will notice that the object, **groundshrub** in our example, will be appended by a number in parentheses to indicate how many copies of the object are in the level. If **groundshrub (0)** were displayed, it would indicate you have not placed any copies of the object into your level. If **groundshrub (319)** was displayed, you would know you had 319 copies of the **groundshrub** object in your level.

Terrain sculpting

In the previous section, you explored how to color and add vegetation to your game world. The result was a flat world with seemingly random colors and ground shrubs. In order to make the environment more realistic, the terrain cannot remain flat. Lumberyard has tremendous tools for modifying the terrain.

Terrain modification tools

To access this tool set, navigate to **RollupBar** | **Terrain** | **Modify**. This brings up the **Modify Terrain** subpanel. Before sculpting our terrain, we will review the associated functionality available on this subpanel:



You will notice that there are three main areas to the **Modify Terrain** pane: **Brush Settings**, **Noise Settings**, and the two **Reposition** checkboxes located at the bottom of the pane.

Brush settings

There are four buttons located at the top of the **Brush Settings** area:

- **Flatten** – This allows you to use the brush to flatten your terrain.
- **Rise/Lower** – Selecting this option allows you to change the height of your terrain. Use the left mouse button to increase the height and *Shift* + left mouse button to lower it.
- **Smooth** – This tool allows you to smooth your terrain. It is recommended that this be performed after all other terrain sculpting operations are completed. You can see the difference in the **Viewport** while you are using this tool.
- **Pick Height** – Selecting this tool presents you with an eye-dropper icon in the **Viewport**. It allows you to detect the height of an existing object. This tool helps you achieve symmetry with your terrain.

Lumberyard provides you with two methods of changing the size of your brush: the **Outside Radius** up/down arrow keys and the slider. You can also synchronize the radius for all brush types. You can set your radius anywhere from `0.01` to `200`. That is a tremendous range, giving you the capability of painting large areas quickly and painting with fine detail.

There are three additional **Brush Settings**. The **Inside Radius** setting determines how flat or round the brush is. **Hardness** and **Height** are the final two options. They provide the functionality their name suggests. The harder the brush, the more defined the edges will be. Consider painting a strip down a wall with a brush. Now, paint a second strip down a wall that has painter's tape on each side of the area you will paint. Once the paint dries and the painter's tape is removed, you will see very hard/defined edges. Fortunately, in Lumberyard, we do not need to wait for the paint to dry.

Noise settings

Noise, as it relates to brush settings, refers to the grainy nature of a brush. The more noise, the grainier the brush will be. A specific setting cannot be recommended because settings are dependent upon what you are painting and what you want to achieve. To fully appreciate this feature and get an idea of what setting works best for your situation, you should experiment with the brush settings.

There are three noise-related settings. First, you can enable or disable noise by using the checkbox. The noise **Scale** ranges from `-100` to `+100`. Lastly, you can set the **Frequency** to a value from `0` to `100`. Frequency refers to how frequently the noise will be applied with the brush.

Reposition

The final area of the **Brush Settings** pane contains two checkboxes: **Reposition Objects** and **Reposition Vegetation**. These features allow the **Lumberyard Editor** to slightly reposition your level's objects and vegetation based on changes you make to the terrain, you can also prevent it from doing so. These settings can save you a lot of time, especially if you are creating a large, open world. If you are placing objects with great precision, you'll likely want to disable these options.

Making terrain modifications

Now that you have a handle on the terrain editing tools, we'll make some modifications to our terrain. With such a large game environment and so much variability, you will likely create something a bit different from what is discussed in this chapter. That is okay. You can experiment while you work through this chapter. At the chapter's end, you can download the incremental build from the book's website. So, do not worry about ruining any of your work. There is great learning value in experimentation.

The following steps will turn the terrain into our game world:

1. Orient your Viewport so you can see the entire terrain. You'll quickly appreciate how large our level is.
2. Select **RollupBar** | **Terrain** | **Layer Painter**.
3. Increase the brush radius to somewhere around `800–900`.
4. Next, select **soil** and paint it on the entire terrain. This will be our base coat, so to speak.
5. Before we paint the **grass** and **rocks**, and add vegetation, let's create some hills and mountains. As you'll remember from the last section, select **RollupBar** | **Terrain** | **Modify** to access the tools.
6. Using the **Rise/Lower** button, paint hills and mountain ridges on the terrain.
7. To add some trees, select **RollupBar** | **Terrain** | **Vegetation** and click the **Add Vegetation Object** button.
8. Navigate to **objects** | **styletown** | **natural** | **vegetation**, select one of the trees, and click the **Open** button.
9. Back in the **RollupBar** pane, select the tree you just added.
10. Select the **Paint Objects** button.
11. In the Viewport, paint the trees where you would like them.

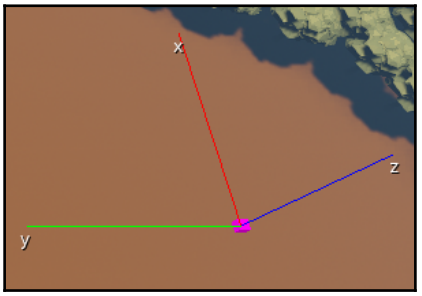
You are not limited to the textures and objects that come with Lumberyard. You can also import trees and other vegetation objects that you purchase or create externally to Lumberyard.

Adding a water feature

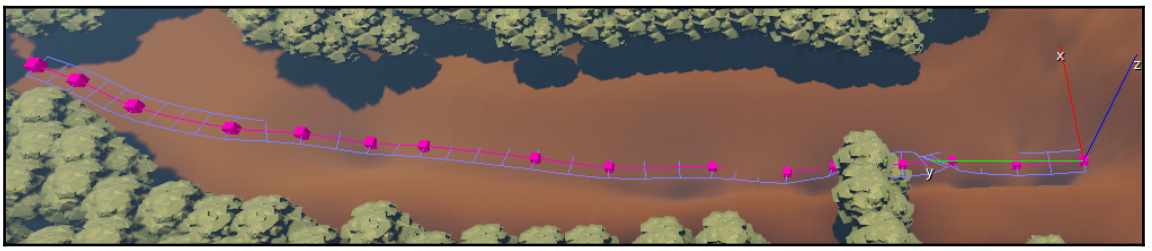
Our game world would not be complete if it did not have water. After all, you added a lot of vegetation in the last section, so let's ensure they have a nice water source. Before jumping into this next task, you should consider where you want the water in your world. Do you want puddles, ponds, lakes, rivers, or an ocean? Planning this out before you start working on it will save you time and frustration later on.

For our game, we will create a river that runs through our game world. Follow the steps below to add water to your level:

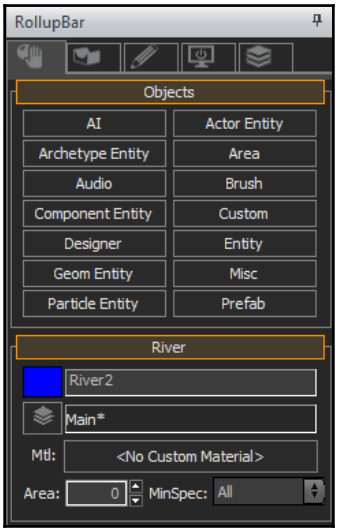
1. Navigate to **RollupBar** | **Terrain** | **Modify** | **Rise/Lower**.
2. Set the **Outside Radius** to `10`. This defines the width of your river.
3. Set the **Height** to `-2`. This will be the depth of your river.
4. Navigate to **RollupBar** | **Objects** | **Misc** | **River**. This will bring up an *XYZ* placement interface in the Viewport:



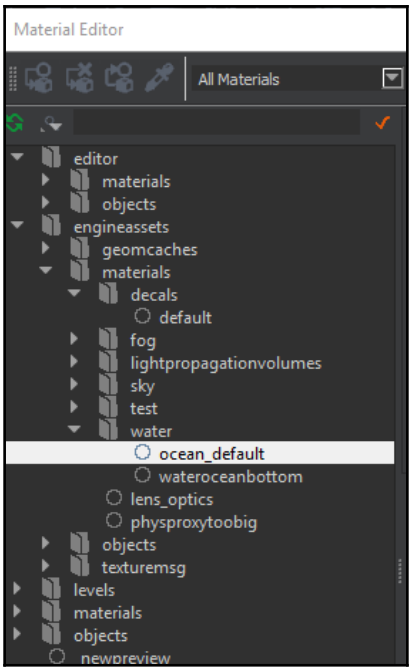
5. In the Viewport, click where you want the river to start, then click in sequence to identify the river's path. Double-click to indicate the river's end point. When completed, you will have a visual representation of your river's path in the Viewport:



6. Next we need to perform a little bit of landscaping on our terrain. In the **River Parameters**, click the **Align Height Map** button. This makes the necessary modifications to your terrain height so that it matches the river's path.
7. So far, we've created the river's path and modified the terrain to match. Our next task is to assign a material to the river. You should still have the **River** object visible in the **RollupBar**. If not, navigate to **RollupBar** | **Objects** | **Misc** | **River**.
8. Click the **<No Custom Material>** area of the subpanel. This will bring up the **Material Editor**:



9. In the **Material Editor**, navigate to an appropriate material. Alternatively, you can upload your own water material:



10. Click the **Add Item to Selected Objects** button. It is the leftmost button on the **Material Editor** toolbar:

11. Close the **Material Editor** interface.

As you saw in both the **Material Editor** and the **River Parameters** sections, there are a lot of settings and adjustments that can be manipulated to make your water feature work exactly the way you want it to. Experimenting with these settings on a test level is recommended. Rivers and other water features can be a bit tricky to get right. Taking the time to experiment with them in a risk-free setting is best. Risk-free, in this context, refers to working in a manner that cannot negatively impact other work.

At the end of this chapter, we will run our game in Game Mode so we can see how our world looks. But before that, we need to play **Mother Nature**.

Playing Mother Nature

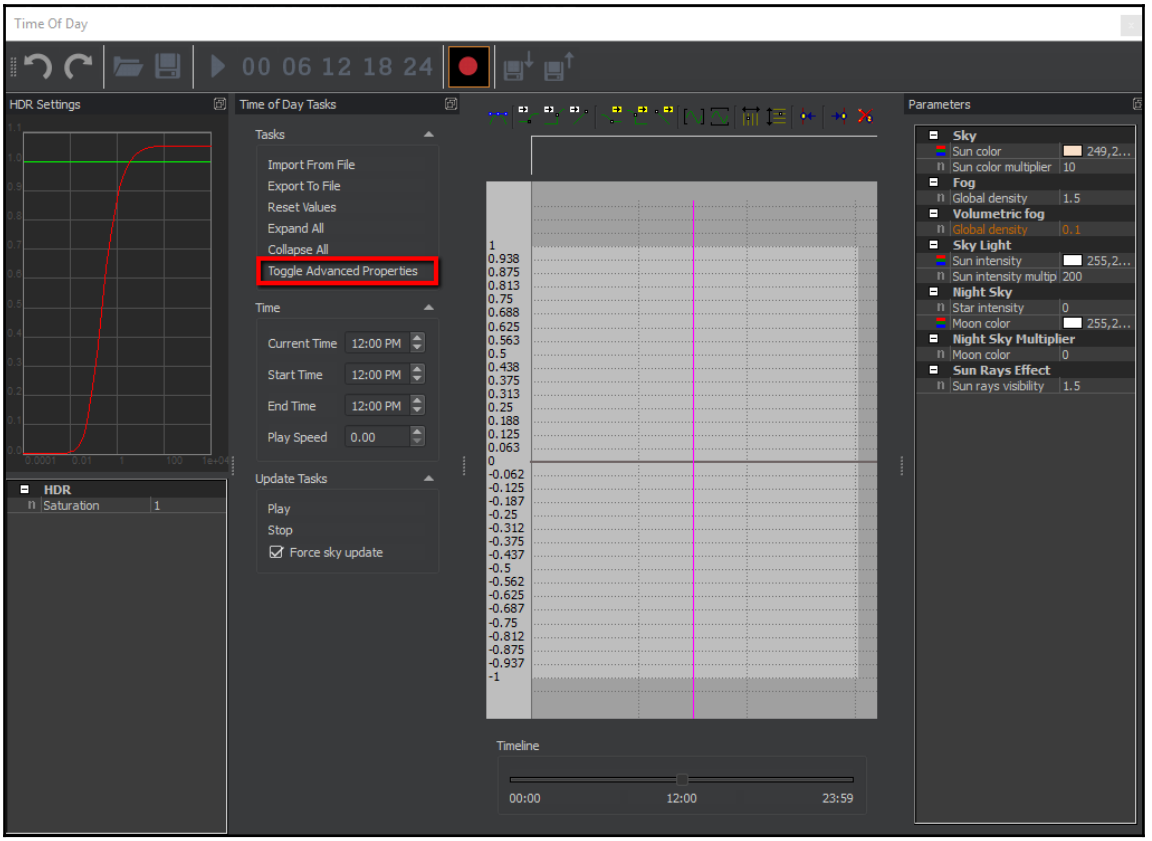
In order to create an immersive game experience for our players, we will add some environmental effects to make the world seem more real. In this section, we will add fog, shadows, and sunlight.

Adding fog

There are several things we can do with fog. We can add it to the entire world, certain areas, and just above water. We will use *Global Fog* using the **Time of Day** functionality.

Using the pull-down menus, select **Terrain** | **Time of Day**. This evokes the **Time of Day** editor:

1. Under **Time of Day Tasks**, click the **Toggle Advanced Properties** link:



2. You will now see a host of settings listed under **Parameters** in the rightmost column. You can experiment with these in a test level. For now, accept all the options and close the **Time of Day** editor.

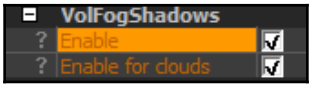
You can learn more about fog in the Lumberyard documentation: `http://docs.aws.amazon.com/lumberyard/latest/userguide/weather-fog-volumes.html`.

Adding shadows

We know that sources of light can cast shadows. The source can be the sun, a moon, or a man-made object such as a light or flashlight.

First, let's add shadows to our fog:

1. Navigate to **RollupBar** | **Terrain** | **Environment**.
2. Under **VolFogShadows**, click both checkboxes:



Yes, it was that easy!

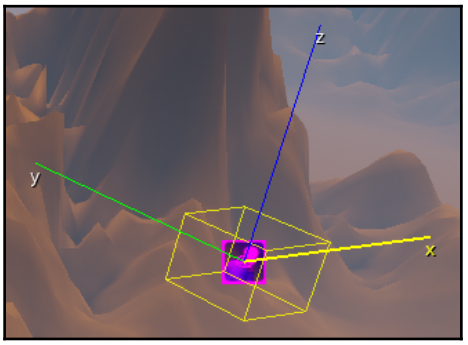
Adding sunlight

Next, we will review how sunlight is managed. Our primary interface for sunlight is the **Time of Day** editor. Toggling the **Advanced Properties** link reveals several properties that can be changed. These properties are under the **Sky** and **Sky Light** categories. There are additional settings for **Night Sky** and **Sun Rays Effect**.

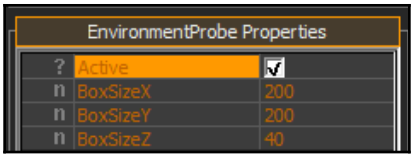
Another way to control ambient light is to add a `Probe`. Let's do that now:

1. Navigate to **RollupBar** | **Objects** | **Misc**.
2. Under the **Object Type** section, select **EnvironmentProbe**. This reveals the parameters associated with the `Probe`.

3. In the Viewport, click where you would like to place the `Probe`. You can change the location any time, so exact placement is not important. The `Probe` is represented by a square object with *XYZ* handles:



4. With the `Probe` selected, turn your focus to the **RollupBar**. You will see an **EnvironmentProbe Properties** section. Check the **Active** checkbox and change the **BoxSizeX**, **BoxSizeY**, and **BoxSizeZ** settings to `200`, `200`, and `40` respectively. When completed, your parameters should match what is shown in the following screenshot:



5. Under the **Probe Functions** section, click the **Generate Cubemap** button. You should now be able to see shadows cast based on this light source.
6. If you want to move your `Probe`, you can follow these steps:

    1. Ensure the `Probe` is selected in the Viewport.

    2. From the pulldown menu, select **Modify** | **Transform Mode** | **Move**. You will see a set of handles on each axis (*X*, *Y*, and *Z*).

    3. Click on the axis handle you want to move, and drag along that axis until the object is where you want it.

You can also move the Probe with direct keyboard entry in the footer area of the Viewport. You will see values for each axis.

Another method of generating light in your game world is to add a light. Here is the process to add a light to your game world:

1. Navigate to **RollupBar** | **Objects** | **Entity** | **Lights**.
2. Click on **Light** and drag it to your level in the Viewport.
3. Modify the placement and size to your liking.

This type of light source is referred to as a *Light Entity*. You have access to a whole host of parameters. Experimenting with each of these parameters will help you become well versed in lighting:

1. In the **RollupBar**, navigate to **Entity Properties** | **Projector** | **Texture** and click the file folder icon.
2. Navigate to **textures** | **lights**. Select a texture file of your choosing.
3. Click the **Open** button to finalize your selection and close the **Preview** window.
4. Next, we will rotate the light. From the pulldown menu, select **Modify** | **Transform Mode** | **Rotate**.
5. In the Viewport, click on the white circle and use your mouse to rotate the light so it points downward. You can zoom in to get a closer look at the direction in which your light is pointing.

Testing your environment using Game Mode

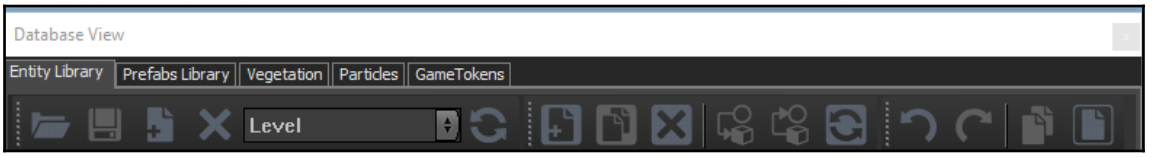
So far, we have accomplished a lot and have several features in our game environment. We have examined our world through the Viewport. The real test is to see and experience our world as a player would when playing our game.

In order to test our game, we will need to add a camera. We'll do that in the next section.

Adding a camera

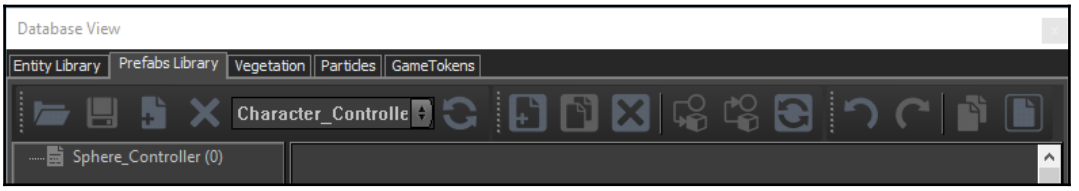
In this section, we will add and configure a camera. Cameras in games are how the player sees the game world. Think of the lens of the camera as the player's eyes. Let's start working on our camera:

1. Using the pulldown menu, select **View** | **Open View Pane** | **Database View**. This will result in the **Database View** window being displayed. As you can see in the following screenshot, there are tabs for **Entity Library**, **Prefabs Library**, **Vegetation**, **Particles**, and **GameTokens**. There are several icons displayed:



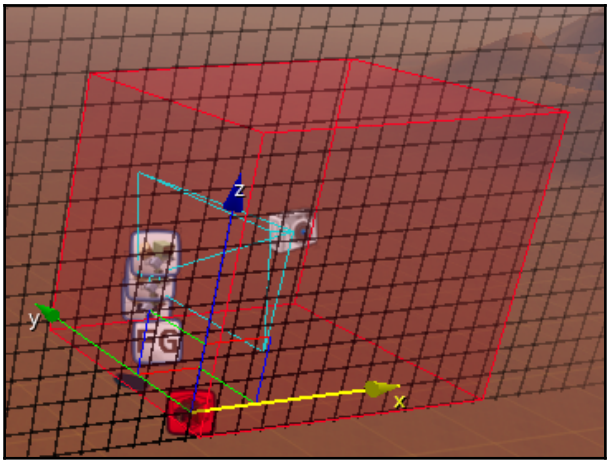
The preceding screenshot shows the **Entity Library** tab. The **Database View** interface changes with each of the five tabs.

2. In the **Database View** dialog window, select the **Prefabs Library** tab.
3. Click the **Load library** icon. It is the first one on the left in the **Database View** dialog window. This displays a list of available prefabs that we can use in our game.
4. Select the `character_controllers.xml` file and click the **OK** button. A character controller is typically used for first- or third-person player control.
5. If necessary, click the expand arrow. You will now see a **Sphere_Controller** listed in the **Prefabs Library** tab of the **Database View** interface. Drag the **Sphere_Controller** to your level in the Viewport. Click the expand arrow, if necessary:



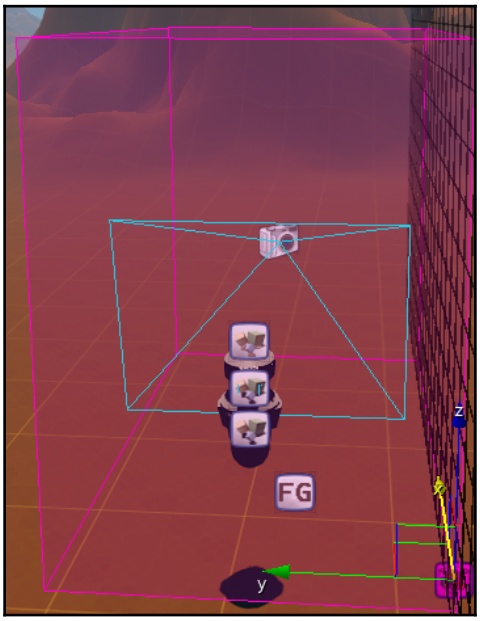
6. Close the **Database View** window.

You will now see the player controller in your level. There is a lot bundled with the player controller. Most notable is the camera. Recall that we said the camera was analogous to the player's eyes? The placement and orientation of the camera is critical, as those settings determine where the player starts the level and in which direction they are looking. The next few steps illustrate how to change those settings:

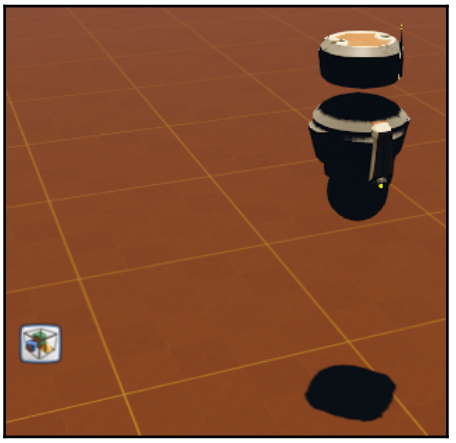


A closer look at our prefab object reveals that it came bundled with several components. First, there is the robot character that comes with Lumberyard. It has a third-person camera. Third-person cameras are ones that are viewed from outside the character.

There are also input controls that will allow the player to control the player-character:



7. Click any open space in your level to deselect the player-controller. As you can see in the following screenshot, the player-character is displayed with shadows and the control cube. It is also too high off the ground. The robot has a roller at its base and that should be on the ground when the level starts:



8. Select the player controller.
9. Use the pull-down menu to select **Modify** | **Transform Mode** | **Move**.
10. In the Viewport, modify the *Z*-axis so the controller is on the surface of your game world.

In the next section, we will test our level and get the player controller mobile.

Game Mode

One of the great features about most 3D game engines is that you can easily switch between development and Game Mode to test the level you are working on. Before switching to game mode, you should consider saving your work. You can accomplish this quickly via the Ctrl + S keyboard combination or by selecting **File** | **Save** from the pull-down menu.

When game mode is evoked, the game becomes playable in the Viewport. There are two methods of switching to game mode. First, you can use the *Ctrl* + G keyboard combination. Alternatively, you can use the pulldown menu to select **Game** | **Switch to Game**.

To exit game mode, simply press the Esc key.

Controls

The player-character already has functionality associated with it. The input controls are listed in the following table:

Input	**Action**
W-key	Move forward
S-key	Move backward
A-key	Move left
D-key	More right
Mouse	Rotate
Spacebar	Jump/elevate

You are now able to explore your game world from the player's perspective.

Summary

In this chapter, we created our game world and made several additions and modifications to help make the game world come alive. Our goal was to create an immersive environment that our players will appreciate. Specifically, we applied colored textures to our terrain, added vegetation, and used sculpting to create hills and mountains. We also created lights and shadows using the **Time of Day** editor. We added a river to give our world an added bit of realism.

Our final accomplishment in the chapter was to test our game world in Game Mode. To enable this, we created a player controller with a camera and input controls. After putting our level in Game Mode, we were able to control the player-character and travel in our game world.

In the next chapter, we will focus on 3D characters. So far, we have only used the robot character that came with Lumberyard. We will look at two other options for player-characters. First, we will create our own 3D character using Lumberyard's character tool, *Geppetto*. We will explore that tool in great detail. We will also look at how to import 3D characters created with external tools such as Blender, Maya, and 3DS.

4

Creating 3D Characters

In the previous chapter, we created our game world. Our game world, also referred to as our game environment, is where the game takes place. It is the environment in which our players will interact. In order for our players to play our game, they need a game character. Our game world is three-dimensional (3D), so our characters will need to be 3D as well.

In this chapter, we will add 3D characters to our game. These characters will come in two forms, player-controlled characters and non-player-controlled characters.

After reading this chapter, you will:

- Understand the concept of 3D characters
- Understand Lumberyard's capabilities regarding 3D characters
- Learn how to use the FBX Importer
- Become familiar with the Geppetto user interface
- Understand how to use Geppetto to manipulate 3D characters
- Learn how to create your own character definition
- Understand the significance of using third-party modeling software

Dissecting 3D characters

Characters are a key part of our game. Characters can come in many forms, including humanoids, animals, robots, and more. We can allow our players to play as any type of character that we instantiate in the game. As you would expect, characters are incredibly more complex than most other game objects, such as a building, rock, weapon, or lamppost.

3D Character vocabulary

Working with 3D characters takes a certain level of understanding. Creating these characters is a complex process and has an associated vocabulary. Here are some of the basic terms used when working with 3D characters:

- **Polygon**: You'll remember from geometry class that a polygon is a plane figure with three or more straight line segments. Polygons can be very complex. They are always closed and contain no curves. When using 3D modeling software, polygons are also referred to as *faces*.
- **Polycount** (short for polygon count): This is an important concept to consider. The polycount is the number of triangles it takes to construct your 3D model. You will see the importance of polycounts in the *Render* section.
- **Level of Detail**: The greater the number of polygons, the greater the level of detail your model will have. In order to reduce render times, you should consider what elements of detail need to be part of the 3D model and which ones can simply be included in the texture.

There are also **Level of Detail** (**LOD**) models. This is when you use multiple models of the same object in a game, each with a different level of detail. Think about a game where the player stands on the coast, looking at the horizon. A ship that is 12 miles away does not need to have the same LOD as when it is just a few yards away.

- **Render time** – The time it takes, in our context, for a game engine to draw a 3D model is referred to as render time. When animated movies or movies with CGI scenes are created, each 3D model has exacting detail and large polycounts. This is why CGI scenes in movies look so incredibly real. Watch the latest dinosaur movie to see this in action. Where games differ from movies is when the rendering occurs. For movies, they are rendered when the project is completed and then distributed. So, the viewer has the final product to watch. With games, the rendering occurs in real time. This is known as **Real-Time-Rendering** and sometimes **Run-Time-Rendering**. A single frame of a movie could take hours to render. With approximately 30 frames per second, the amount of time it takes to render an entire movie is enormous. We do not have this luxury in games. If games rendered scenes as slowly as movies do, no one would play games. For these reasons, we need to keep our polycounts low and properly manage our 3D character's LOD.
- **Texture** – A texture is a graphic pattern or representation. In games, we use textures to paint our game world and its objects. We used textures, referenced as *materials*, in `Chapter 3`, *Constructing an Immersive 3D Game World*, when we

created our game environment. A *Tiled Texture* is a special kind of texture file used in games. When we select a material in Lumberyard to paint our grounds, we might use a dirt texture tile (or tiled texture). These tiles fit seamlessly together on all sides. This saves us a lot of time, especially when dealing with large areas.

- **Topology** – This is a reference to how a mesh is fitted over a model. For example, imagine stretching a rubber bag over an irregularly shaped pumpkin. The final shape of the rubber bag is the topology.
- **Vertex** – A vertex is a single point in 3D space. The plural form of vertex is vertices.
- **Edge** – An edge is a line that connects two vertices. When you create 3D characters and other 3D models, it is best to ensure you only use triangle polygons. The game engine will convert any non-triangle polygons to triangles by adding edges. This can result in a higher polycount than if you imported your 3D character with only triangle polygons.
- **Mesh** – A mesh is a group of polygons that form a portion of or an entire 3D model. As you can see in the following screenshot, the mesh is for the 3D player character we will use in our game. It is represented in 2D and was generated by software, so it is not something that you will need to manually create. Creating this 2D mesh is accomplished through mathematical computations. It is similar to peeling an orange so that the skin is flat:

The process

Creating a 3D character from scratch takes considerable effort and cannot be done entirely using Lumberyard on its own. The process starts with a 2D design or perhaps a photograph. From there, sculpting can occur to help form the 3D model. A re-topology process is used to reshape the model's topography so that the model looks just the way you desire. You next create meshes. Once the model is shaped and skinned the way you want it, you will add a skeletal structure using a rigging process. From there, you can add weighted painting, and apply textures and shading. Finally, you will create animations for the character.

Each step listed above takes time, talent, and knowledge of modeling software. In the next section, you will discover what tasks can be performed in Lumberyard regarding 3D characters.

Lumberyard's capabilities

When we work with characters in Lumberyard, we have multiple goals. For player characters, we want the player to navigate the characters in the game world and we want the character to be animated both by the player and automatically depending on the in-game situation. For non-player characters, we want animations to be evoked through our AI system, sparked by in-game events and real player interactions.

Lumberyard allows us to import characters from Maya and 3D Studio Max. You can also import characters that were exported to FBX format. FBX stands for Filmbox and is an industry standard file format that packages 2D, 3D, audio, video, and animation content for an object.

Lumberyard has two specialized tools for 3D characters: Geppetto, Lumberyard's character tool, and a Mannequin system for character animation. We will look at Geppetto in this chapter and save the Mannequin system for `Chapter 5`, *Animating Your Characters*.

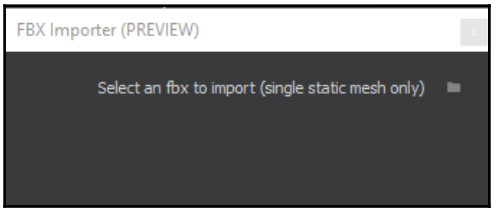
FBX Importer

Lumberyard's **FBX Importer** permits importing 3D characters into your level that were created using external 3D modeling tools such as Autodesk Maya and Autodesk 3DS Max. At the time of this book's publication, the **FBX Importer** feature was still under preview and not completely implemented. The steps listed here may change in the future, but provide you with a general sense on how the feature will function:

1. Launch the Lumberyard Editor.
2. Open the `level_01` file that we have worked on in previous chapters.

You can download the `level_01` file from the book's website.

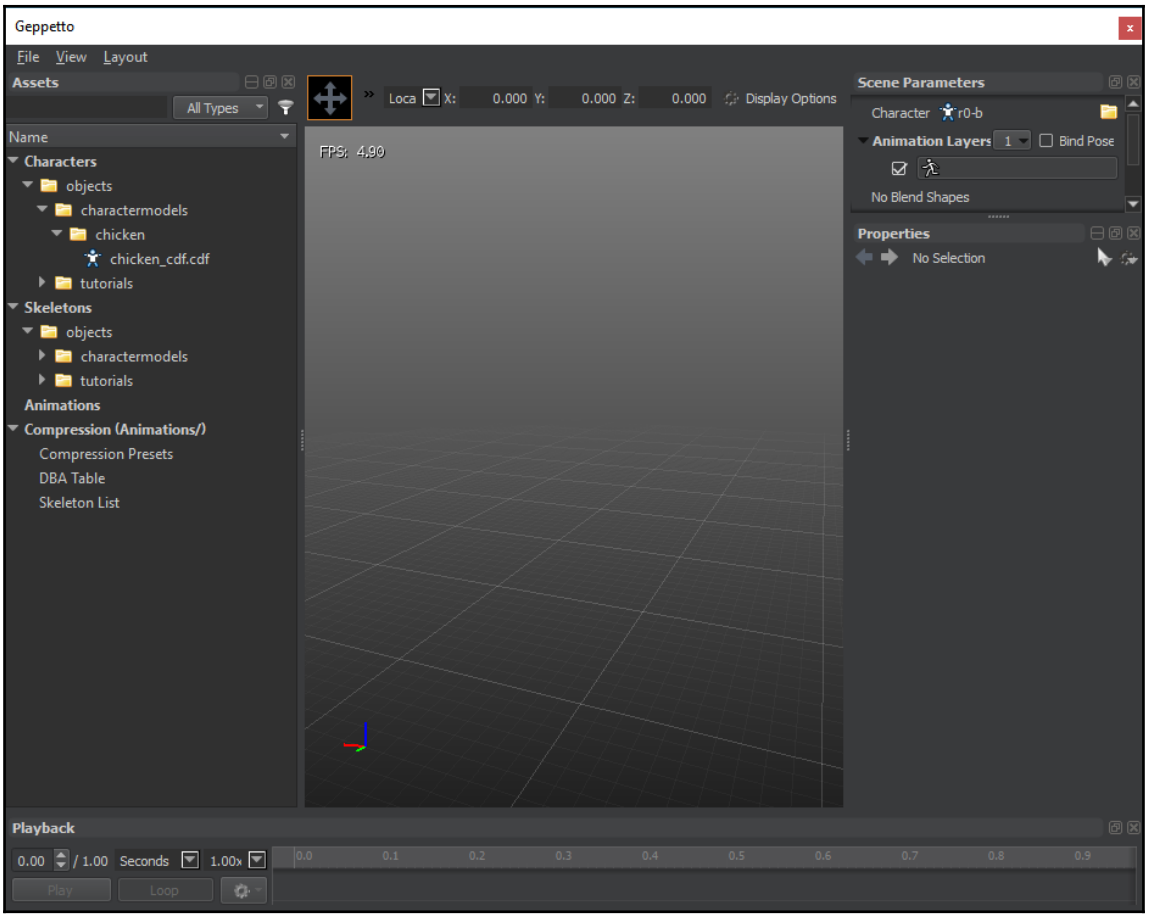
3. From the pull-down menu, select **View** | **Open View Pane** | **FBX Importer**. This will evoke the **FBX Importer** dialog window. At the time of publication, this feature was still in preview, so your interface might look different:



4. Click the file folder icon and navigate to your FBX file. Once you locate and select the FBX file, click the **Open** button.
5. Next, you will see an intermediate screen that lists your filename and additional information. Click the **Import** button.
6. That is all there is to this process. Later in this chapter, we will manipulate our game's 3D characters.

Geppetto

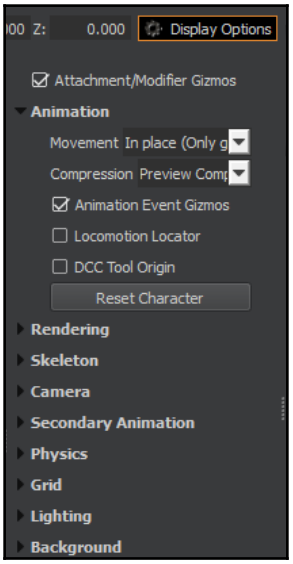
Geppetto is Lumberyard's animation tool. To access Geppetto, select **View** | **Open View Pane** | **Geppetto** from the pull-down menu. As you can see in the following screenshot, the dialog window presents you with significant access and functionality:



At the top of the window is a set of pull-down menus. The **File** menu allows us to open, save, and create new characters. The **View** menu permits us to toggle specific areas of the interface on/off. With the **Layout** menu, we can save any layout configurations we want with the Geppetto interface.

The default layout, as depicted in the preceding screenshot, contains the **Assets**, **Scene Parameters**, **Playback**, and **Properties** areas. You can also view the **Blend Space Preview** and **Animation Event Presets** by accessing the **View** menu.

There is a powerful tool in Geppetto that allows you to control display options. In the top banner area below the pull-down menus, you will see a **Display Options** button with a wheel cog icon. Clicking that button opens a host of options, as illustrated in the following screenshot:



Under **Display Options**, there are nine categorical options that can be expanded to view and change settings:

- **Animation**
- **Rendering**
- **Skeleton**
- **Camera**
- **Secondary Animation**
- **Physics**
- **Grid**
- **Lighting**
- **Background**

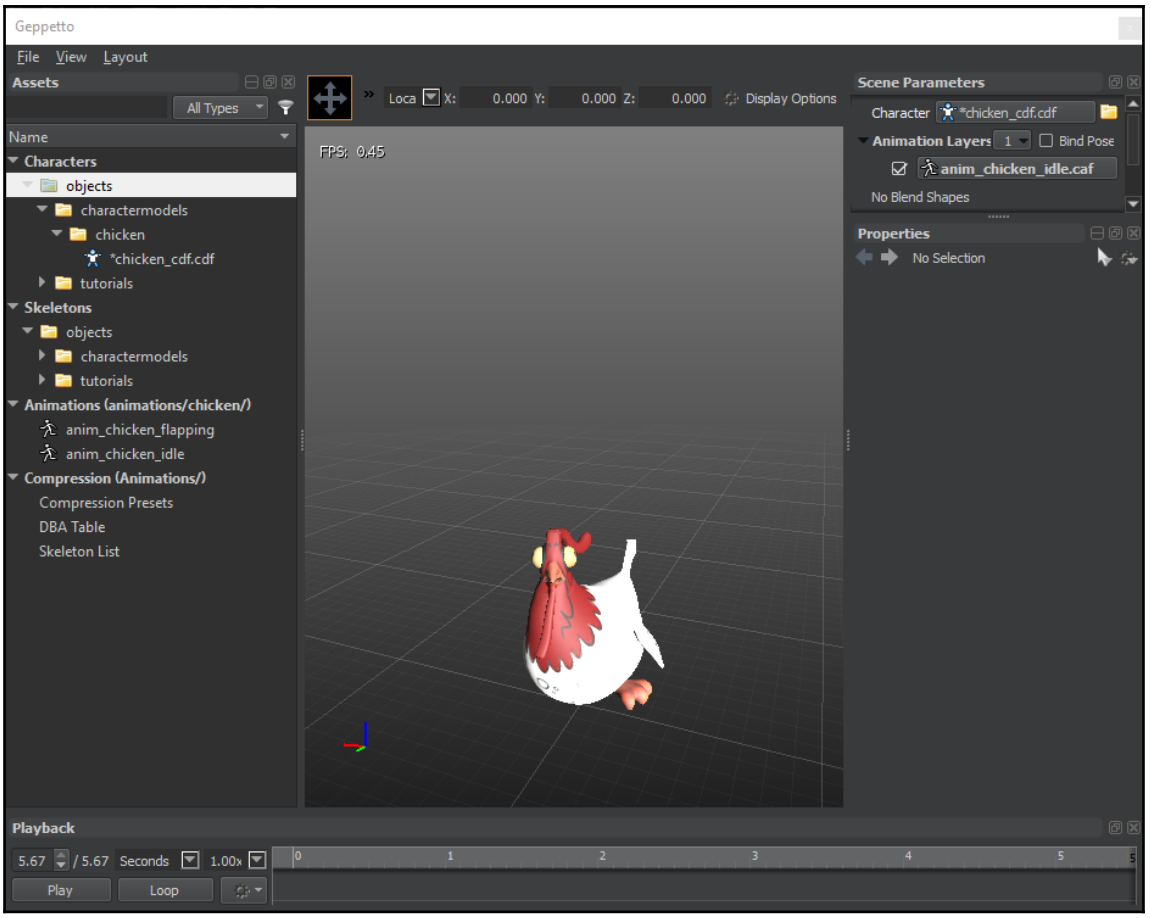
Once you have made all the desired adjustments, clicking the **Display Options** button again will close that pane. To explore Geppetto further, we will examine a sample character. Then, we will create our own.

Exploring Geppetto

In this section, we will load the chicken character that comes with Lumberyard. Once the chicken character is loaded, we will make modifications in order to fully explore Geppetto:

1. Select **View** | **Open View Pane** | **Geppetto** from the pull-down menu to access Geppetto.
2. In the **Scene Parameters** pane, located in the upper-right corner of the interface, click the file folder icon. This will bring up a file dialog window.
3. Navigate to the `objects/charactermodels/chicken` folder and select the `chicken_cdf.cdf` file.
4. Click the **Open** button.
5. You should see the chicken character in Geppetto's viewport. If you do not see the character, pan and zoom as necessary until the chicken is presented as shown in the following screenshot:

Viewport navigation: You can use your mouse's scroll wheel to zoom in/out, the right mouse button to pan, and your keyboard's *WASD* keys to move.



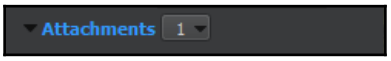
- You can test an animation by selecting **Animations** in the **Assets** pane hierarchy, expanding the hierarchy if necessary, and selecting an animation. Once selected, the animation should play automatically. You can also control playback using the buttons in the **Playback** panel.
- We will cover animations in `Chapter 5`, *Animating Your Characters*.

Attaching objects to characters

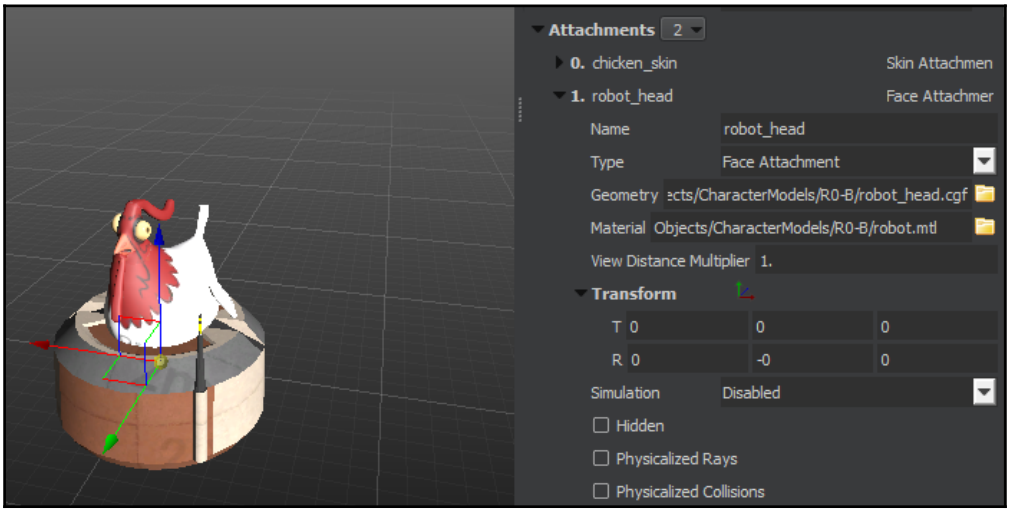
It is common for games to have items attached to characters. Imagine playing a role-playing game where you are searching ancient ruins for gold and weapons. When you find a forged sword, you pick it up and carry it in your predominant hand. In this instance, the sword object was attached to your character.

Attached objects can be very complex and even have their own animations. Follow these steps to attach an object to our chicken:

1. In **Geppetto**, select the `chicken_cdf.cdf` file in the **Assets** hierarchy.
2. In the **Properties** pane, click the down arrow to the right of **Attachments**:



3. In the pop-up dialog, click **Add**. This will expand the **Properties** pane and require additional information about the attachment.
4. In the **Type** field, select **Face Attachment**.
5. Click the folder icon next to the **Geometry** field and navigate to the sample `robot_head.cgf` file. Click the **Open** button to finalize your selection. This selects the character geometry file.
6. Click the folder icon next to the **Material** field and navigate to the sample `robot.mtl` file. Click the **Open** button to finalize your selection. This step selects the material file for the attachment.
7. As you can see in the following screenshot, the robot head is now attached to the chicken character. While chickens do not need robot heads, this exercise illustrates how to attach objects to characters and what asset files you will need:



Creating your own character

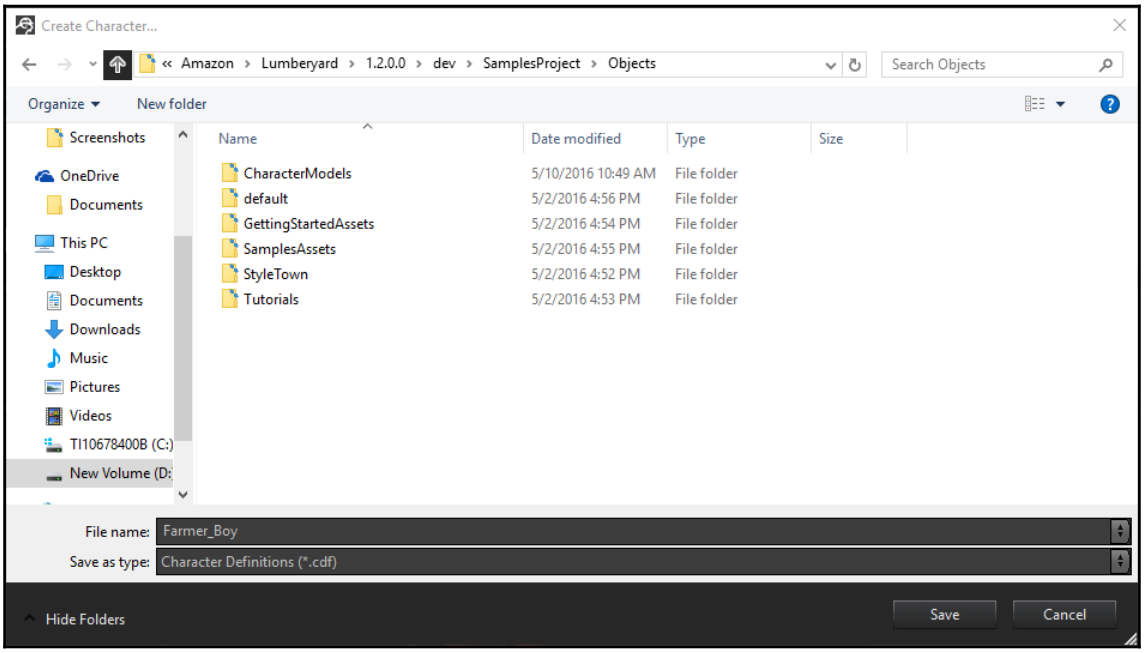
When we create a character using Geppetto, we start by creating a character definition file: a `.cdf` file. An important precursor to creating the character definition file is to have already created, at a minimum, a character skeleton file (`.chr`) and a skinned geometry file (`.skin`). These files can be generated from Autodesk Maya or Autodesk 3DS Max.

For instructions on how to generate the `.chr` and `.skin` files, consult the Autodesk Maya or Autodesk 3DS Max documentation. Additional support is available in the Lumberyard documentation.

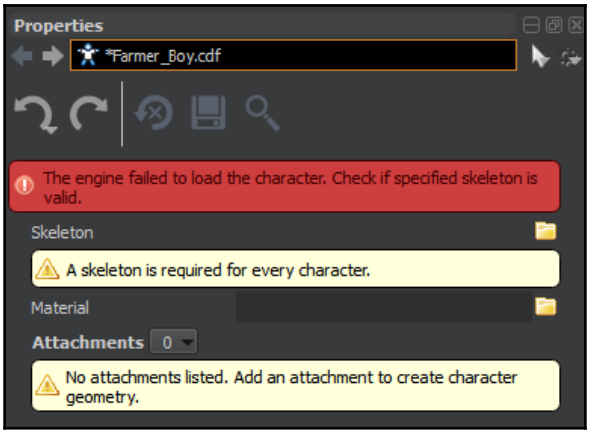
Character definition file

Here are the steps to create a new character definition file using Geppetto:

1. Select **View** | **Open View Pane** | **Geppetto** from the pull-down menu to open **Geppetto**.
2. In the pull-down menu, select **File** | **New Character**. This will open a **Create Character** file dialog window:

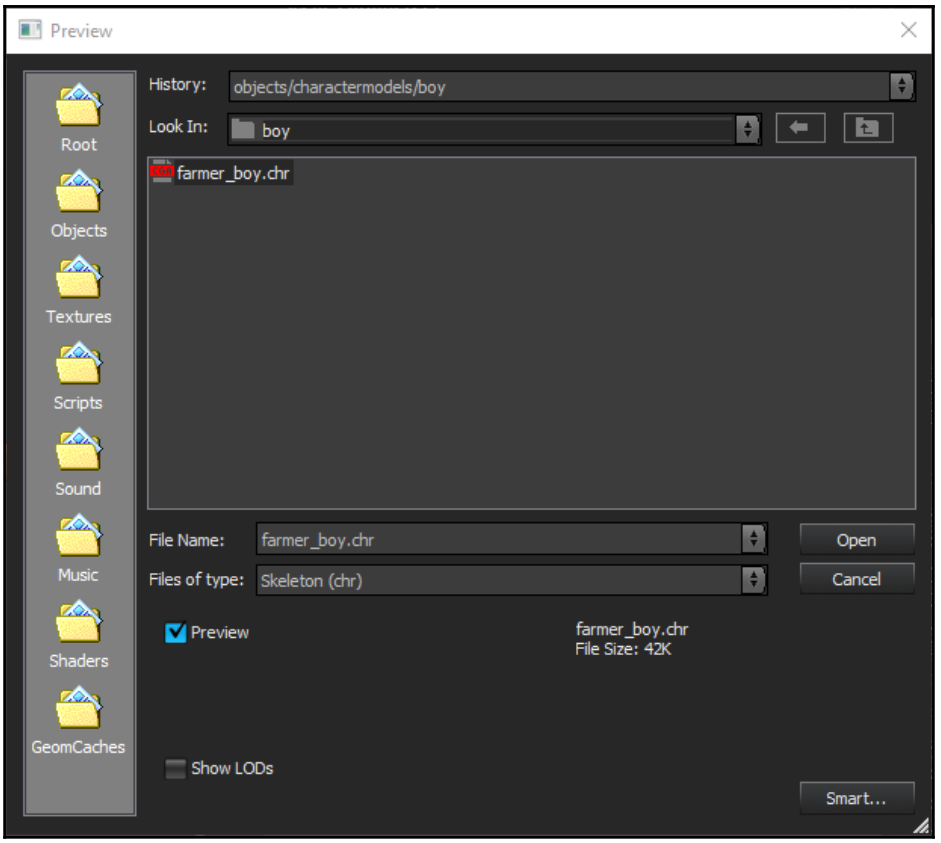


3. Enter `Farmer_Boy` for the filename.
4. Navigate to where you would like the new character definition file to be saved and click the **Save** button. You might be prompted, as shown in the following screenshot, to connect the skeleton and skinned geometry files to the character definition file you just created:

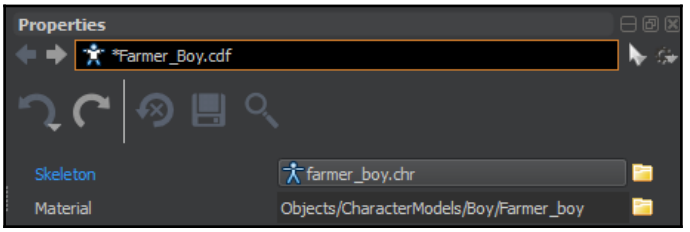


At this point, you may want to download the Farmer Boy character files from this book's website. You can copy the files into an object/character folder so you can easily navigate to the files from within Geppetto.

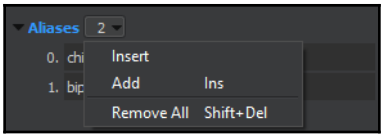
5. In the properties section of the **Geppetto** interface, click the file icon to the right of the **Skeleton** label.
6. Navigate to the `farmer_boy.chr` file provided on this book's website and click the **Open** button:



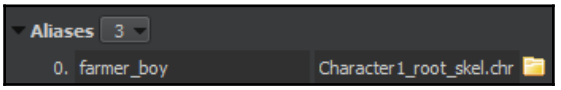
7. In the properties section of the **Geppetto** interface, click the file icon to the right of the **Material** label. Navigate to the location of the `Farmer_boy.mtl` file provided on this book's website and click the **Open** button.
8. You should now have entries for both the **Skeleton** and **Material** items of your character in the **Properties** subpanel:



9. Next, we will add our skeleton to the Geppetto's **Skeleton List**. In the left-hand pane of **Geppetto**, select **Compression (Animations)** | **Skeleton List**.
10. Next, in the **Properties** pane, click the number to the right of **Aliases** and select **Add**:



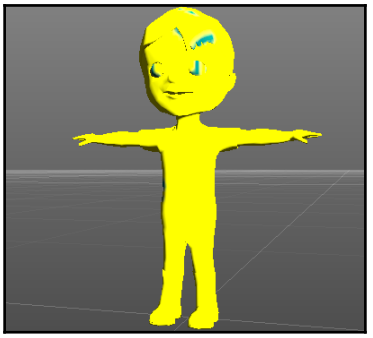
11. Click the file icon to the right of the Alias line you created in the previous step. Navigate to the `Character1_root_skel.chr` file and select **Open**.
12. Rename the alias `farmer_boy` to give us a simpler reference:



13. After creating your 3D character definition in Geppetto and attaching the skin and skeleton to the character, you are ready to add attachments. We will do that in the next section.

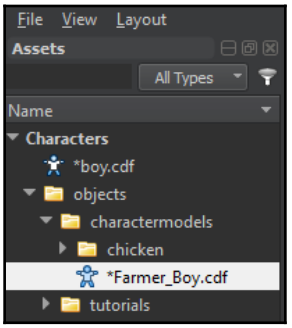
Adding attachments

At this point, your character looks like a 3D yellow model of a boy:



To apply the skin, hair, and clothing, we need to add an attachment. Here are the steps to accomplish that:

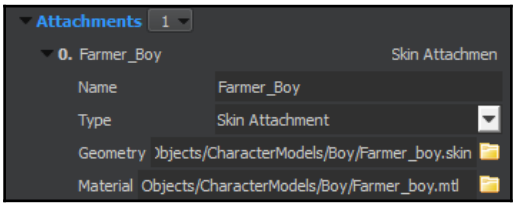
1. In **Geppetto**, select your 3D character from the hierarchy pane:



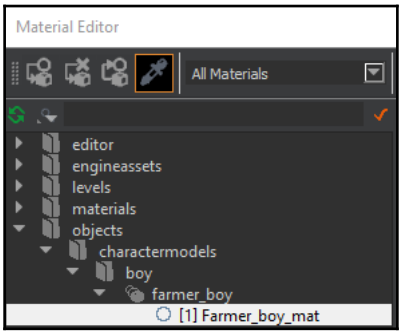
2. In the **Properties** pane, click the number to the right of **Attachments** and select **Add**. This will add an attachment to your character. Next we will define that attachment.
3. Expand the new attachment in the **Properties** pane to reveal its properties.
4. Name the attachment `Farmer_Boy`.
5. Use the pulldown menu to the right of the **Type** field to select **Skin Attachment**. This defines the type of attachment:

6. Select the file folder icon to the right of the **Geometry** field.
7. Navigate to the `Farmer_boy.skin` file and click the **Open** button.
8. Select the file folder icon to the right of the **Material** field.

9. Navigate to the `Farmer_boy.mtl` file and click the **Open** button:



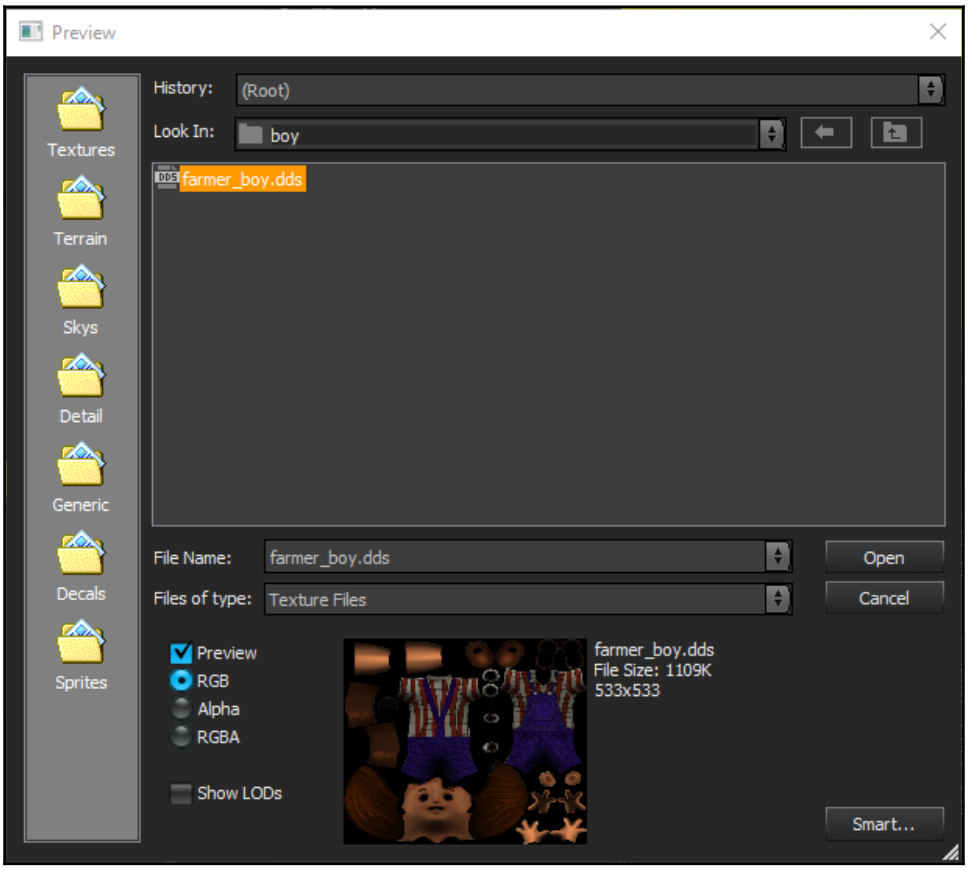
10. With **Geppetto** still open and your 3D character selected and visible in the viewport, open the **Material Editor**. To accomplish this, select **View** | **Open View Pane** | **Material Editor** from the pull-down menu.
11. In the Material Editor, select **Farmer_boy.mat** in the hierarchy list:



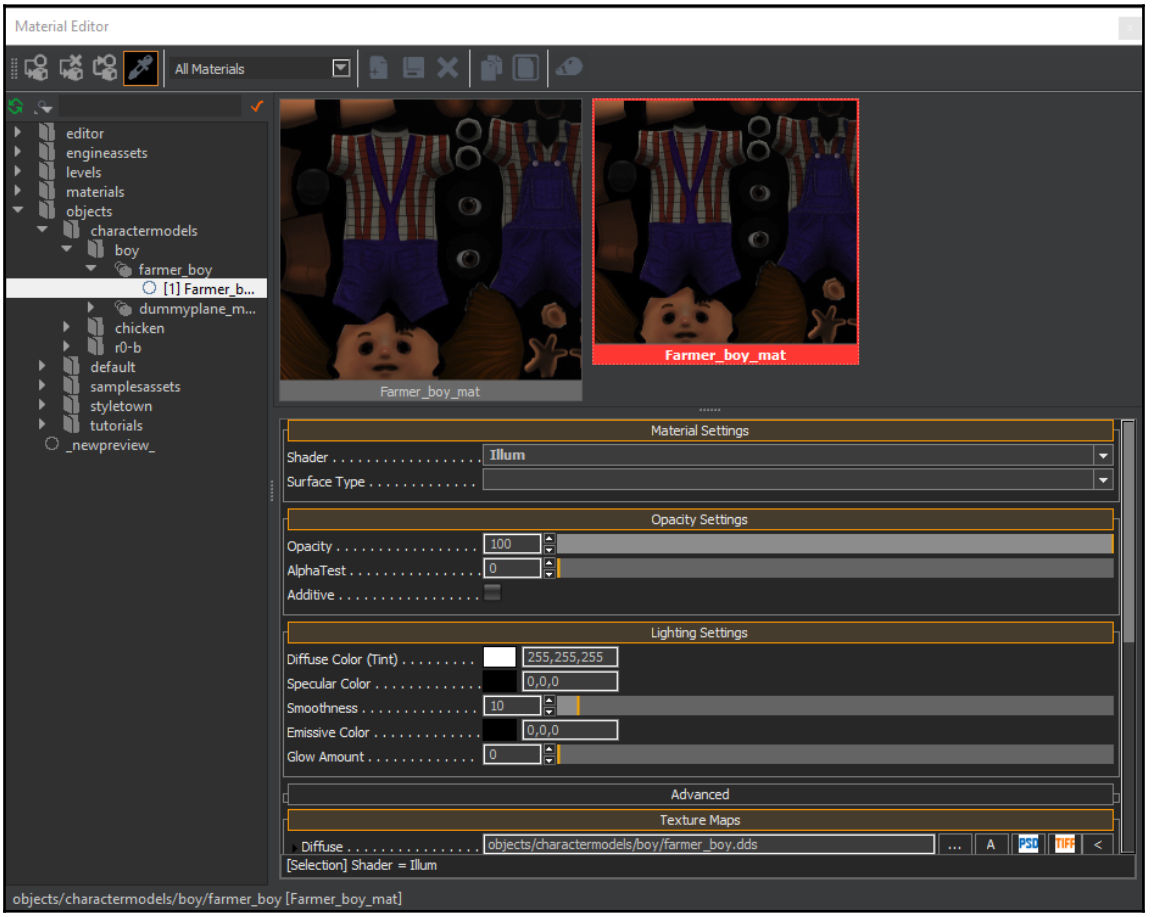
12. Still in the **Material Editor**, scroll in the **Properties** pane to the **Advanced** section. Select the **ellipsis** (**…**) button to the right of the **Diffuse** field.

As Lumberyard continues to evolve, some functions are apt to be reorganized in the user interface. If you cannot find **Diffuse**, try looking under **Texture Maps**.

13. Navigate to the `farmer_boy.dds` file. You will see a preview of the character material file. Click the **Open** button:

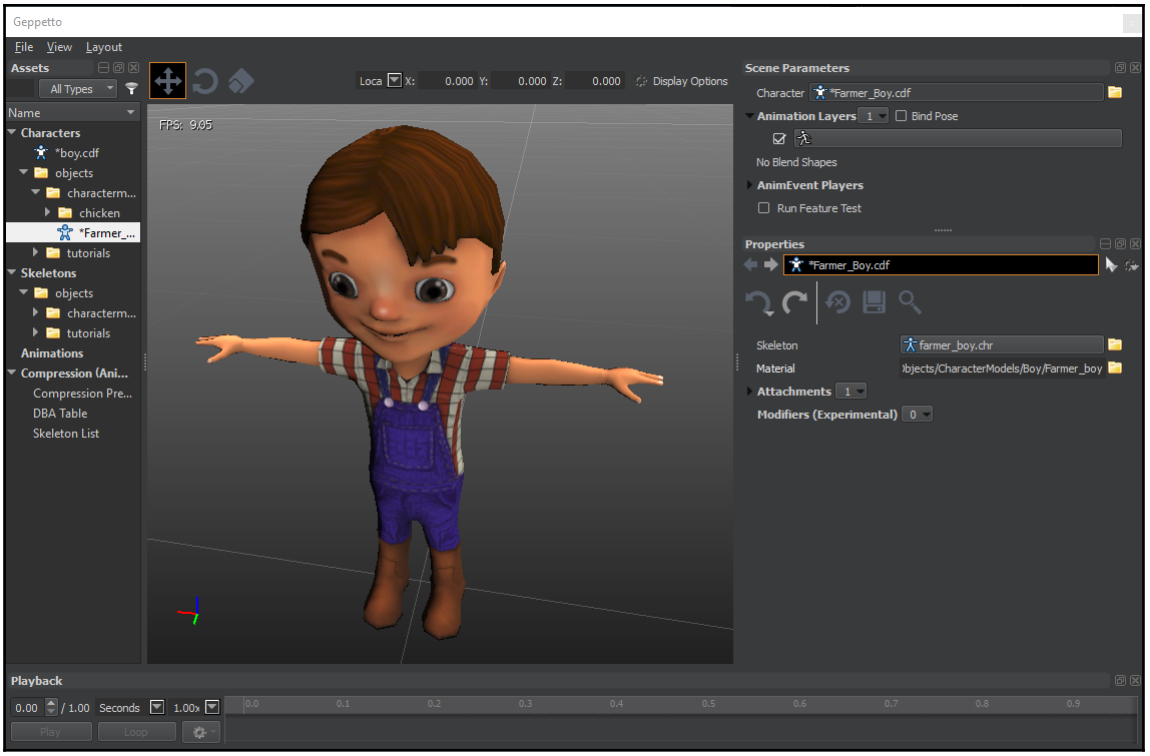


14. You should now see the material thumbnail images above the properties pane. Click the first icon in the **Material Editor** toolbar. When you hover over the icon, you will see that it is the **Assign Item to Selected Objects** button:



15. Close the **Material Editor** and return to **Geppetto**.
16. You will now see the 3D character displayed in full color in Viewport of **Geppetto**.

17. Save your work:



Summary

In this chapter, we discovered the complexities of 3D characters in Lumberyard. Other than the included sample characters (a robot and a chicken), we need to use third-party software such as Autodesk Maya or Autodesk 3DS Max to generate the skeleton and skin files. We explored Geppetto, an important component of Lumberyard. Geppetto is Lumberyard's character tool. We used files provided from the book's website to create our own 3D character. Our character, a farm boy, was created using Geppetto and the **Material Editor**.

In the next chapter, we will take an in-depth look at character animations. Specifically, we will look at sample animations and learn how to create animations of our own. The next chapter will also give an overview of Mannequin, an important subset of Lumberyard.

5
Animating Your Characters

In the previous chapter, we looked at how characters are created in Lumberyard and how they can be imported from other digital content creation tools. We also took an in-depth look at Lumberyard's character tool, Geppetto. In this chapter, we will examine the process of animating our game characters. Our primary tool will be **Mannequin**, Lumberyard's animation tool. This chapter covers Mannequin's user interface and its functionality.

After reading this chapter, you will:

- Understand animation concepts as they apply to 3D games
- Understand the Mannequin filesystem
- Be able to navigate the Mannequin user interface
- Understand the Mannequin workflow
- Understand the basics of Flow Graphs
- Understand how animations are incorporated into Lumberyard games

Basic animation concepts

Animation is more than the art of motion. The 3D games we play are full of motion, not all of it is animation. It is important to differentiate between the two. Motion is when an object moves or is moved. If an apple falls from a tree, it moved from the tree to the ground. This is an example of motion. Animation is when something moves on its own. The apple's descent to the ground did not occur because the apple was alive and decided to fall to its death; rather, the apple was forced to move either by gravity or other means.

When a player throws a hand grenade into a moving vehicle, the vehicle is likely to explode. The motion involved is the hand grenade's flight, and the vehicle's forward motion. The explosion will cause several particles of the vehicle, and people and things in it, to catapult in various directions. This is an example of physics creating motion. We will explore game physics in `Chapter 6`, *Creating Gameplay*. What about the player throwing the hand grenade? That motion was an animation. The character moved on its own or the player controlling the character initiated the animation. Both cases are examples of animation.

So, in our context, animation is associated with characters — player controlled and non-player controlled. In the case of player controlled characters, the animations are initiated by user actions. An example would be using the spacebar to start a jump animation. Non-player characters are animated based on in-game events, controlled by **Artificial Intelligence** (**AI**) scripts. We will look at how to create AI for your Lumberyard games in `Chapter 6`, *Creating Gameplay*.

Before we can animate characters in Lumberyard, we must first ensure our characters have been created with proper skeletal structures. We took care of this in `Chapter 4`, *Creating 3D Characters*.

Introducing Mannequin

Lumberyard's Mannequin system is a key feature of the game engine and one that sets it apart from other game engines. With Mannequin, we can define movement fragments that are part of a set, or family, of movements. Each movement is given a unique identification so that it can be referenced by code and for grouping. As an example, we might have a power drink in our game that, when consumed, our character exhibits new behaviors. These might be longer strides, faster running, shaky hands, and a widened look of the eyes. Each of these can be crafted as movement fragments and grouped together and employed as needed.

This level of specificity allows us to create realistic animations in reaction to in-game behaviors, other characters, and environmental components. Nearly every character has a walk cycle. Imagine if your character had extreme fatigue based on a lack of water consumption and exposure to desert heat. That character might walk a bit differently. Instead of creating an entirely new walk animation sequence, we can incorporate a group of fragments we created that show the effects on the character.

Understanding the Mannequin file conventions

The Mannequin system is very complex and requires several files to make up a system. Most of these files are not created by Mannequin, instead you must create them manually. Lumberyard's documentation provides details on each file type and how to structure their contents. The following table provides an overview of these files:

File Name	File Type/Extension	Remarks
Animation Database	`*.adb`	This important file contains the fragments and the transitions between them.
Character Definition	`*.cdf`	As you'll remember from *Chapter 4*, *Creating 3D Characters*, this file contains references to the character file (`*.chr`) and attachments.
Controller Definition	`*.xml`	Mannequin uses this file to set up a mannequin. It contains references to fragments, tags, and other scope definitions.
Fragment ID Definition	`*.xml`	Fragment IDs are used to group fragments as discussed in the previous section.
Preview Setup	`*.xml`	In this file, we identify our character controller (for the player controlled character), the character, and the animation database.
Sequence	`*.xml`	Stores animation sequences. Mannequin generates this file.
Tag Definition	`*.xml`	We can use tags for later reference in code. Examples include: thirsty, dying, injured, turbo, and anything else your imagination can create.

Mannequin file considerations

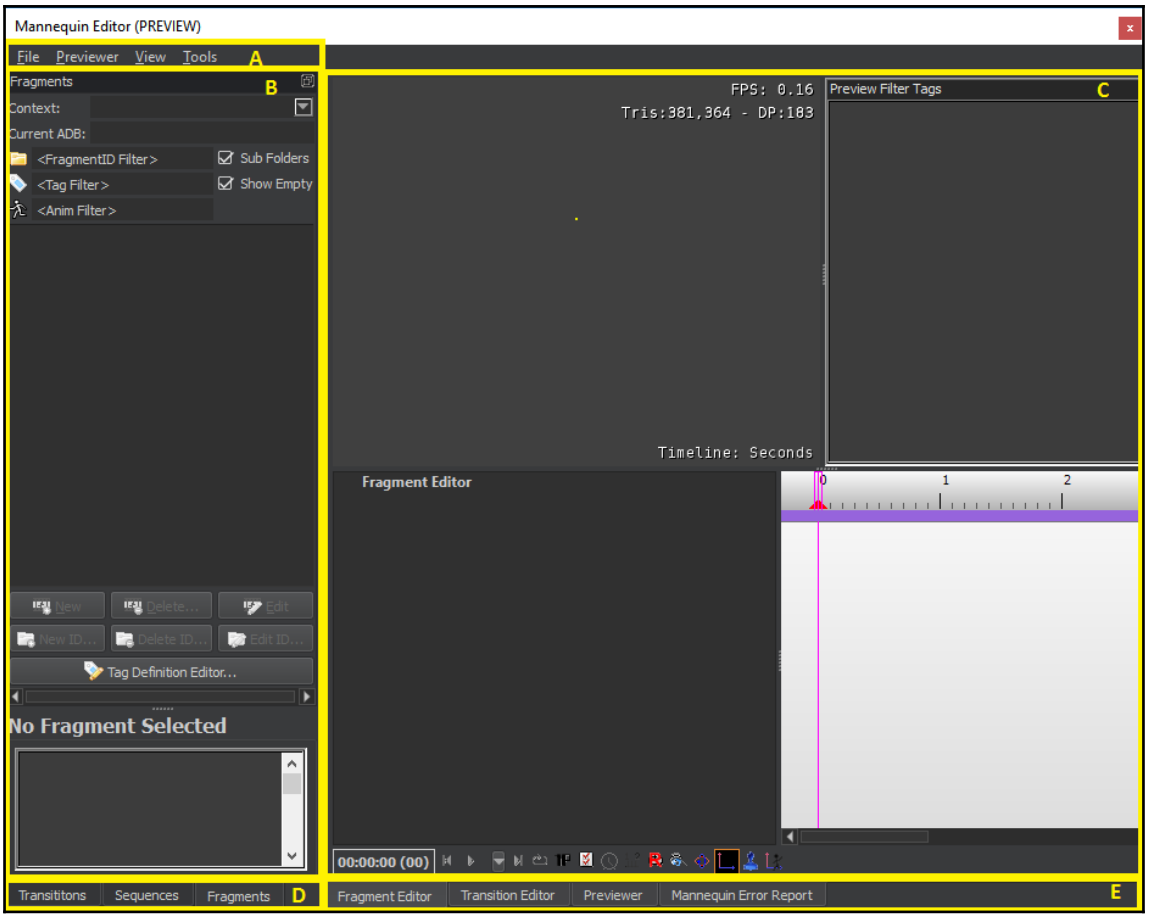
As previously stated, Lumberyard's Mannequin system is very complex, making it easy to inadvertently introduce errors into your game. Here are some important things to keep in mind regarding the files used in the Mannequin system:

- Name your Controller Definition file the same as your character's name. While this is not a requirement, it will save you a tremendous amount of frustration later.
- Name your Preview Setup file to match your character and Controller Definition files.
- Adopt a change-preview-save mindset when working with the animation files. After each change you make, preview it and, if the change produced the results you wanted, save your work. That will ensure you have a fresh save before moving on to the next task.

Before we start creating with Mannequin, let's review the user interface. We will do that in the next section.

Getting familiar with Mannequin's UI

The Mannequin system is accessible using the top menu. Select **View** | **Open View Pane** | **Mannequin Editor**. As you can see in the following screenshot, there are five areas of the **Mannequin** interface:



Area A – pull-down menus

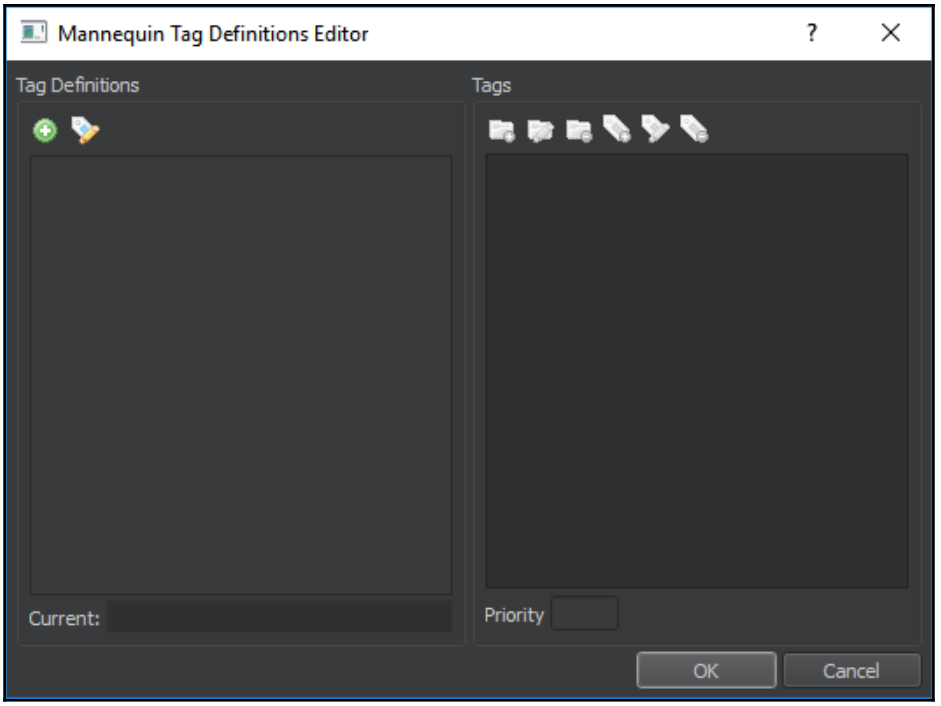
There are four top-level pull-down menus: **File**, **Previewer**, **View**, and **Tools**.

The menu options are not always enabled. They are contextual in nature which means they will only be enabled, or available to you, if they are of use based on the current context of your work.

File menu

The **File** menu provides you with the ability to:

- Load a Preview Setup file. These files are XML files that were discussed in the previous section.
- You can use the **Context Editor** to edit a Preview Setup file.
- The **Animation DB Editor** is used to create and edit an animation database.
- The **Tag Definition Editor** is where you can add, edit, remove, and manage your fragment tags. The green plus icon in the top-left corner of the **Mannequin Tag Definitions Editor** dialog window is how tags are added. Once you have created tags, you can group them as discussed in the previous section:



- You can save your changes using the **File** | **Save Changes** menu option.
- The final option on the File menu is to re-export all modified files.

Previewer menu

The **Previewer** menu option provides three basic functions regarding mannequin sequences. We can create a new sequence, load an existing sequence, and save a sequence that we made changes to using the **Mannequin Editor**.

View menu

The **View** menu has four items listed that you can check or uncheck. When an item is checked, it is displayed at the bottom of the interface as an **Editor Pane Tab.** By default all four editor tabs are enabled. The listed items are:

- **Fragment Editor**
- **Previewer**
- **Transition Editor**
- **Error Report**

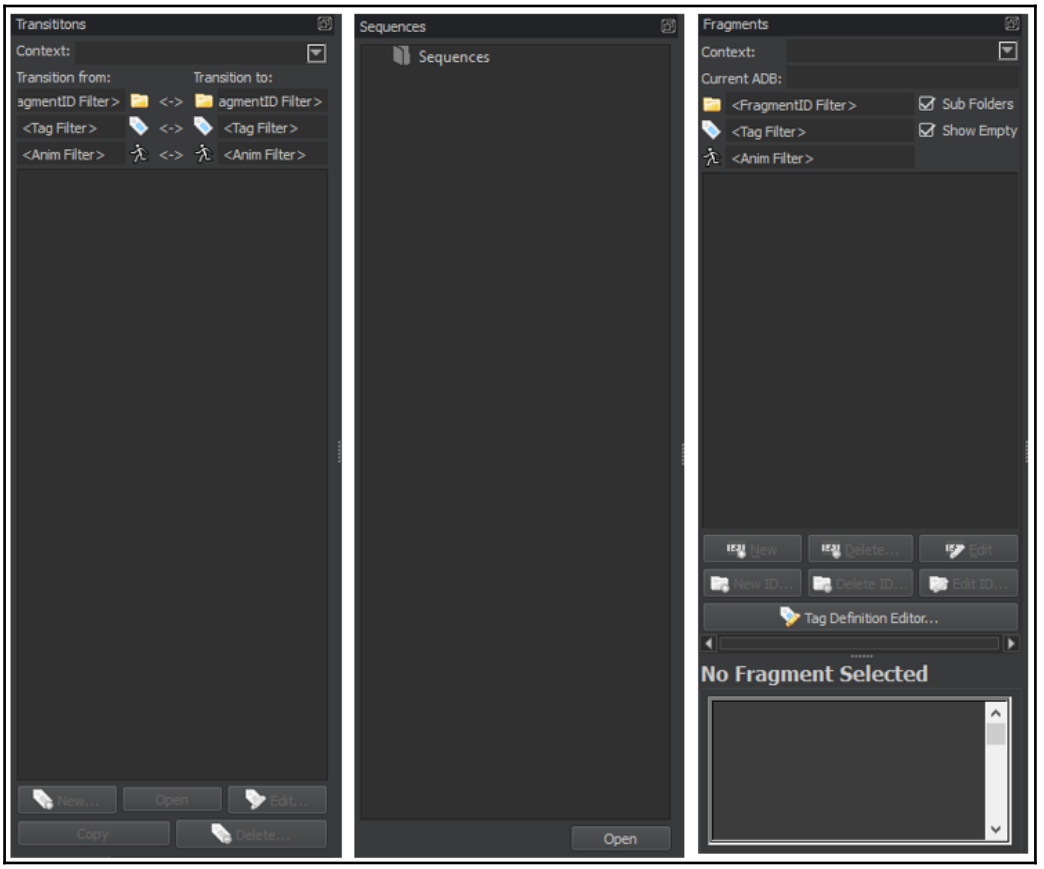
We will look at these later in this chapter.

Tools menu

The **Tools** menu permits two functions. The first function is to save a list of all used animations. The second function is to save a list of all used animations based on the current preview.

Area B – Browser pane

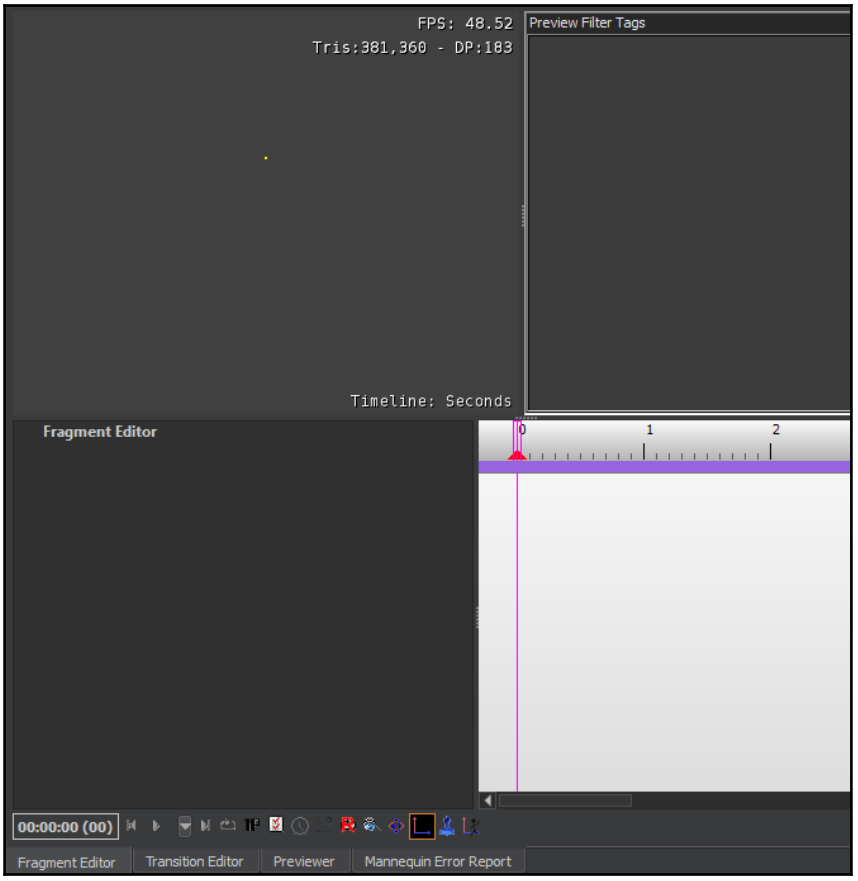
The **Browser Pane** is the left-most area of the Mannequin dialog window located directly beneath the pull-down menus. The contents of this pane are determined by which **Browser Tab** is selected. This pane can display the **Transition**, **Sequences**, and **Fragments Browsers**. The following screenshot illustrates the individual browser contents and layout:



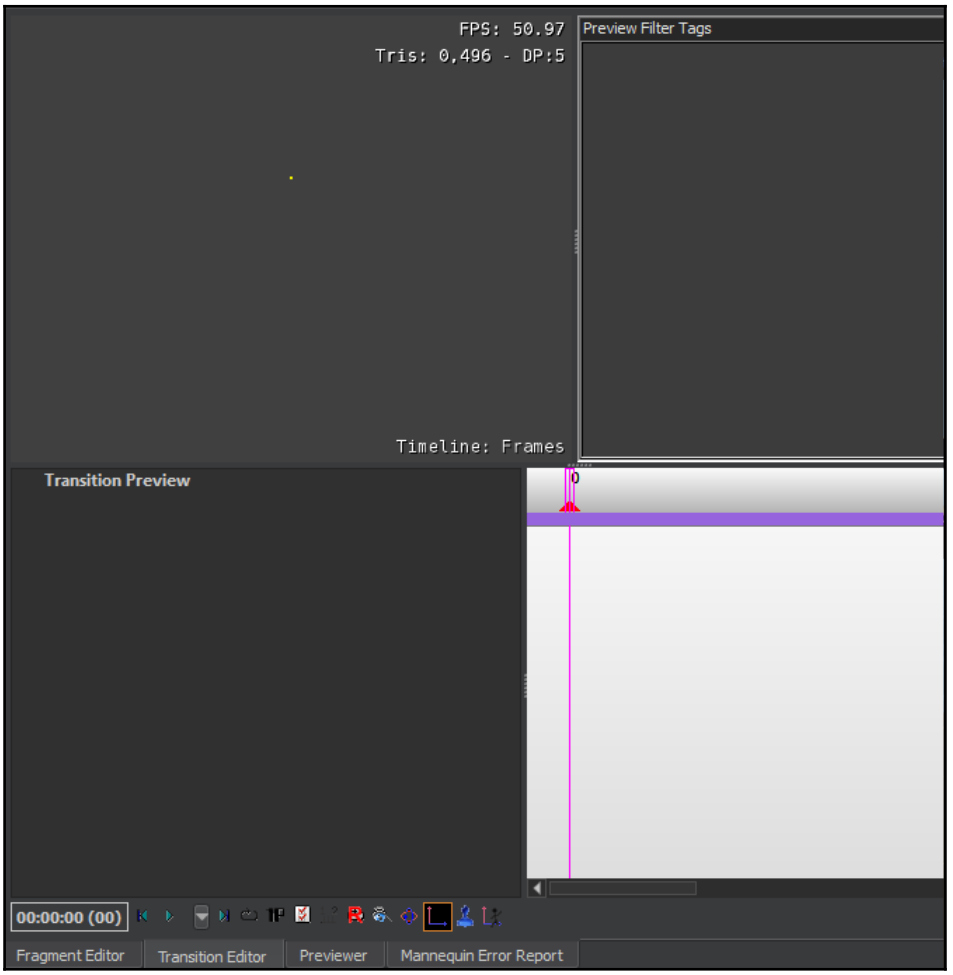
Area C – Editor pane

The large pane that takes up most of the horizontal and vertical space in the dialog window is the **Editor Pane**. The contents in that pane are dependent upon which **Editor Tab** is selected.

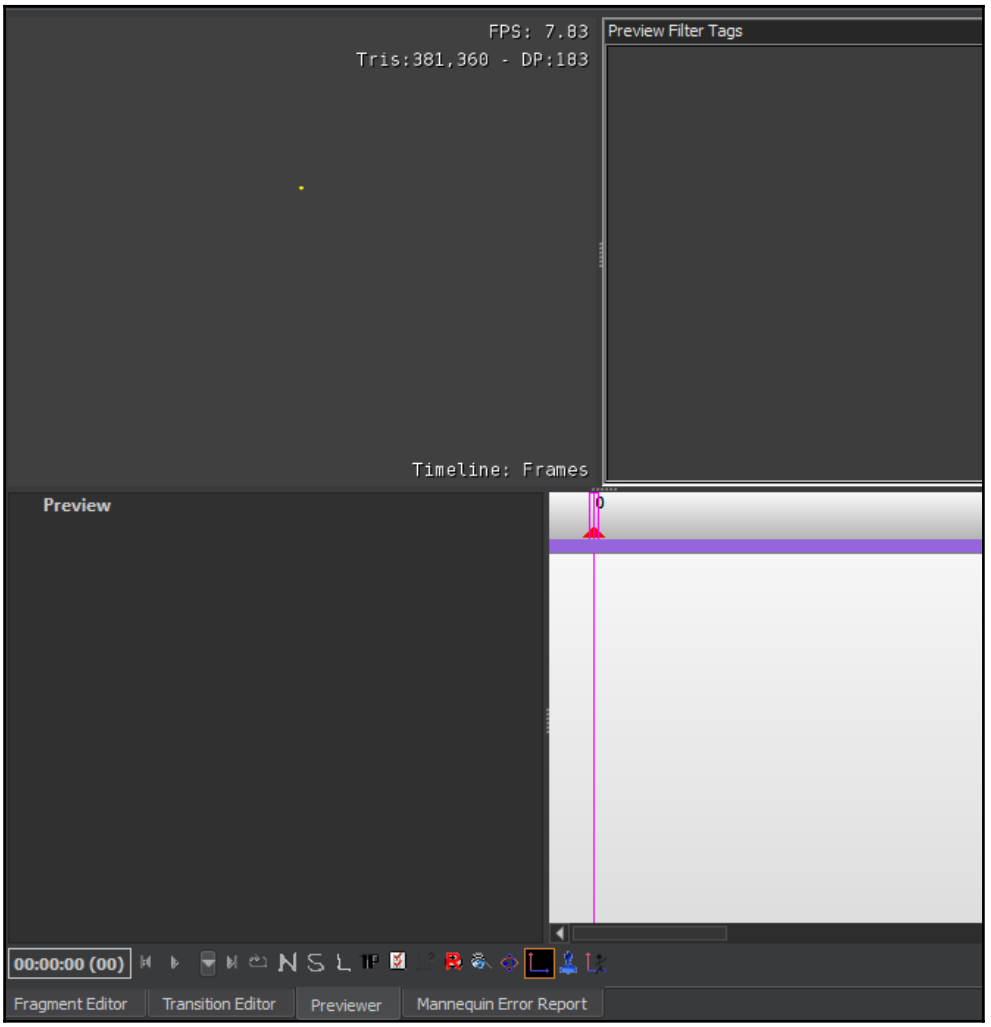
When the **Fragment Editor** is selected, the **Editor Pane** displays five areas. The content consists of timelines, editor, tags, and controls:



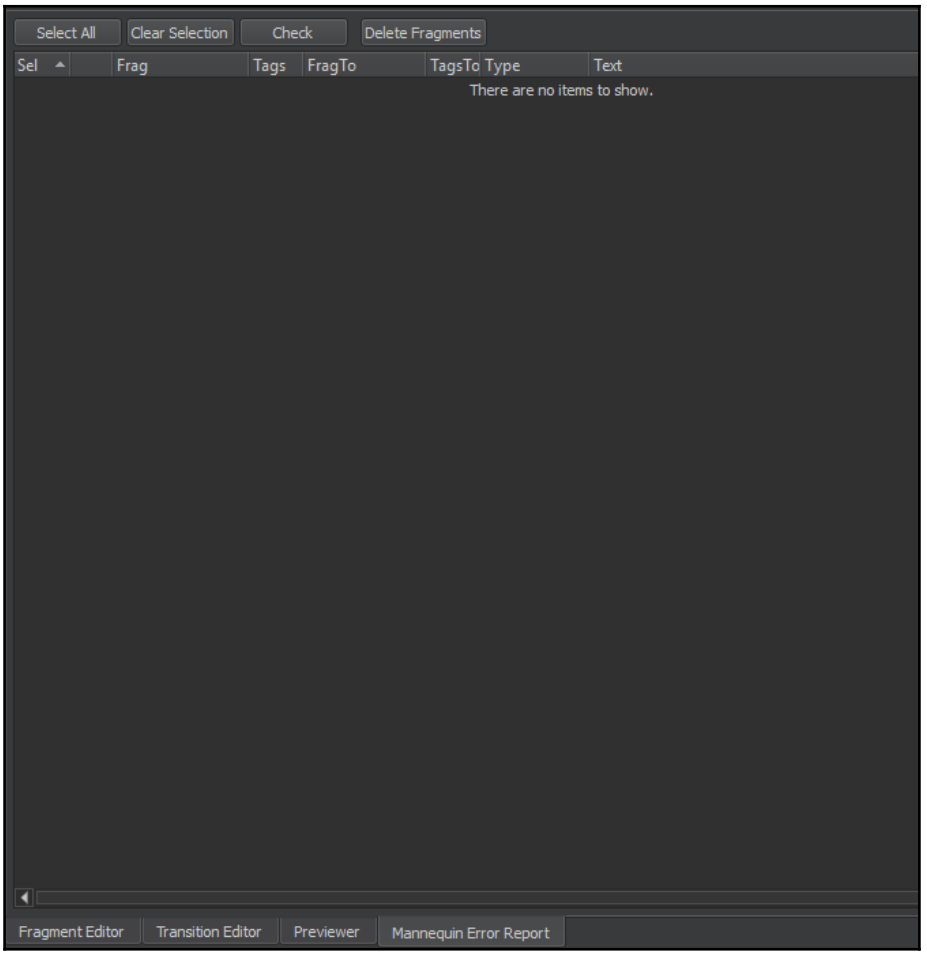
When the **Transition Editor** is selected, the **Editor Pane** displays five areas. The content consists of timelines, editor, tags, and controls:



When the **Previewer** is selected, the **Editor Pane** displays five areas. The content consists of timelines, preview, tags, and controls:



When the **Mannequin Error Report** is selected, the **Editor Pane** displays a list of errors that can be reviewed for debugging:



Area D – Browser Tabs

The **Fragments Browser Tab** is selected by default. You can easily switch between the **Transitions**, **Sequences**, and **Fragments Browsers** by selecting from these three tabs.

Area E – Editor Tabs

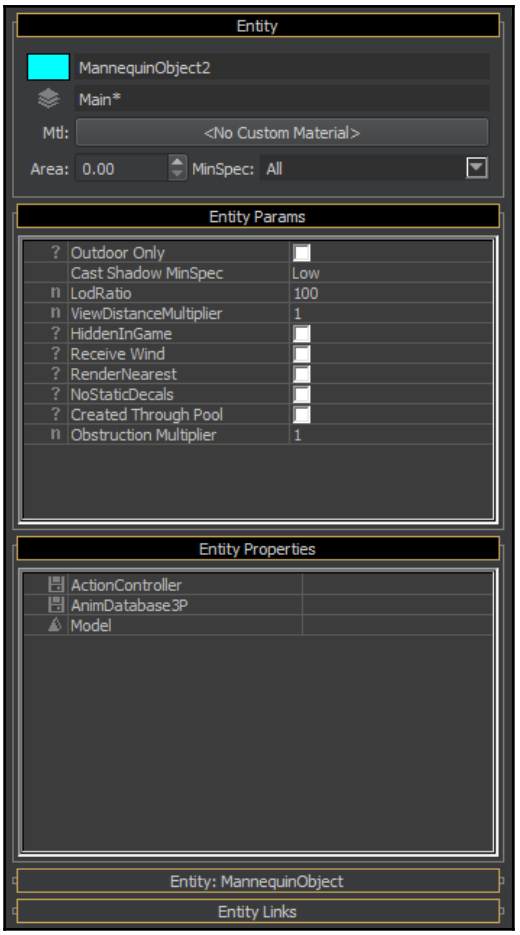
There are four **Editor Tabs** at the bottom of the **Mannequin** dialog window: **Fragment Editor**, **Transition Editor**, **Previewer**, and **Mannequin Error Report**. Each tab will cause the contents of the **Editor Pane** to change. In addition, when the **Fragmentation Editor** and **Transition Editor** are selected, the corresponding **Browser Pane** will also be displayed.

Using Mannequin

Before you can use Mannequin, you must create a **Mannequin Entity**. This process includes identifying the character model, the action controller, and the animation database. To access this functionality, you will select **Entity** | **Anim** | **MannequinObject** from the **RollupBar**.

As you can see from the following screenshot, you will next perform the following actions:

- Name the Mannequin Object
- Assign a material
- Edit, as necessary, the entity's parameters
- Assign the entity's properties:
    - Controller
    - Character model
    - Animation database



All about fragments

One of the keys to mastering Mannequin is to understand fragments and how they are used. Fragments, as previously mentioned, are snippets of motion also known as animation clips, or simply clips.

Transitions can be created between fragments so that the animations seem more natural. For example, when a person transitions from a walk to a run, they do not stop walking then start running. The transition between the two states (walking and running) relies on fragment identifications (Fragment ID) and tags. We previously looked at these components, and reiterate them here as a closing to the Mannequin system to indicate their significance.

As of this book's publication date, Lumberyard's **Mannequin Editor** was still in preview. Once the public release is issued, functionality, workflows, and user interface components might be different than what is presented in this chapter.

Adding animations to characters

Using the **Mannequin Editor** provides great control and fidelity with character animations. With Lumberyard, we have options other than just using the **Mannequin Editor**. We can also use Geppetto, Lumberyard's character tool.

In `Chapter 4`, *Creating 3D Characters*, we created a farmer boy character and applied skin and clothing material to it. Next, we will breathe life into it with animations.

Animation files

To animate characters, you will first use a third-party modeling and animation tool such as Autodesk Maya or Autodesk 3DS Max. Those tools can export animations to compressed **Intermediate Character Animation Files**. These files have a `*.i_caf` file extension.

You can download the intermediate character animation files from the book's website. Here is a table with the filename and associated animation description:

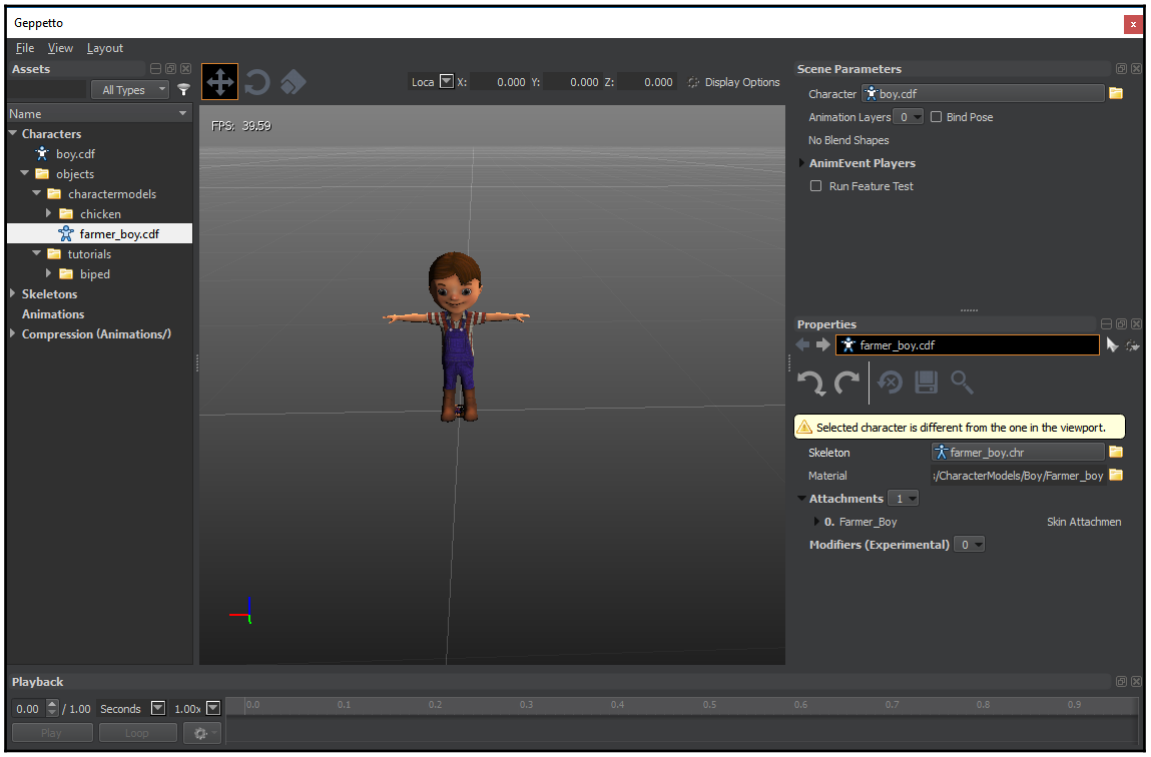
Filename	**Animation Description**
`Boy_Idle.i_caf`	The boy standing in one place with minor movements of legs, hands, and face.
`Boy_Run.i_caf`	The boy running.
`Boy_Take.i_caf`	The boy bending over to take an object from the ground.
`Boy_Talk.i_caf`	The boy standing in one place, talking.
`Boy_Walk.i_caf`	The boy walking at a normal pace.

These are intermediate files because Geppetto will compress them into Character Animation files with a `*caf` file extension for use in Lumberyard.

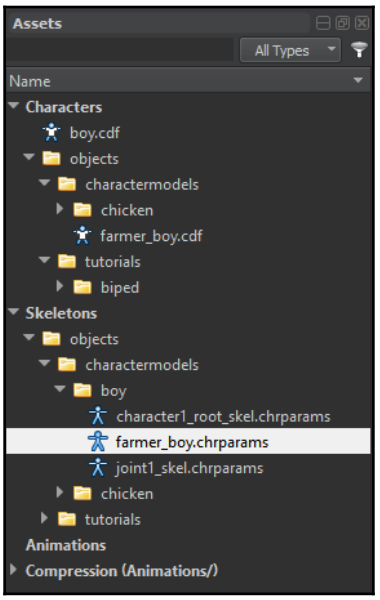
Importing intermediate Character Animation files

To import our intermediate Character Animation files into Lumberyard, we will use Geppetto. First, we will load Geppetto and get it ready for our work. Here are the set up instructions:

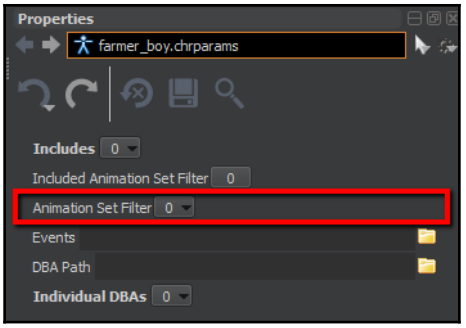
1. Open Geppetto by selecting the **View** | **Open View Pane** | **Geppetto** option from the top menu.
2. Select the farmer boy character in the **Assets** pane.
3. Using your mouse and keyboard, zoom in on your Viewport so the character is well within view:



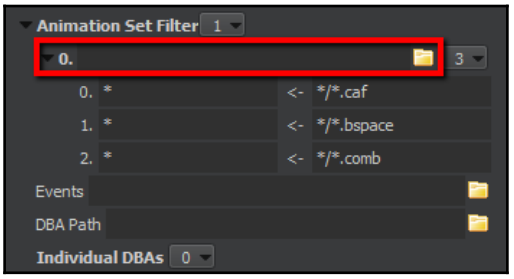
4. In the **Skeletons** section of the **Assets** pane, select the **Character Parameters File** for the farmer boy character:



5. With the **Character Parameters File** for the farmer boy character still selected, click on the down arrow box to the right of **Animation Set Filter** in the **Properties** pane. This will result in a drop-down menu with three options. Select **Add**:

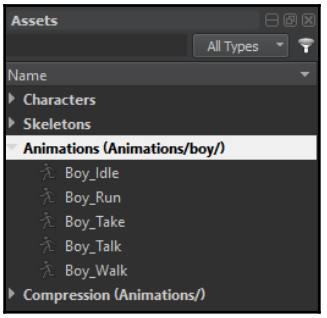


6. Click on the file folder icon to the right of the **Animation Set Filter** you just added:

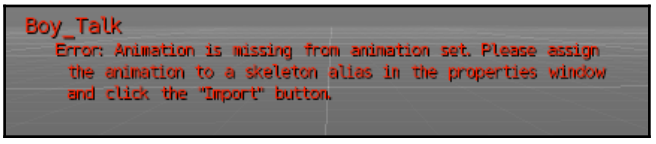


Navigate to the file folder containing the intermediate Character Animation files. These files will have a file extension of `*.i_caf`. You can download the intermediate Character Animation files from the book's website.

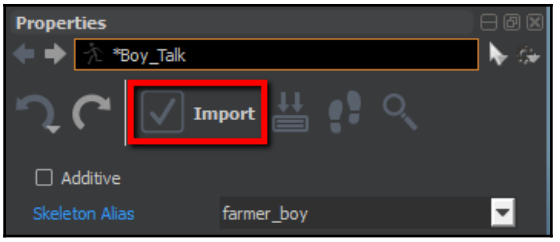
7. Once you have located the file containing the intermediate Character Animation files, click the **Select Folder** button.
8. Back in the **Animations** section of the **Assets** pane, you will see the **Run**, **Take**, **Talk**, and **Walk** animations listed:



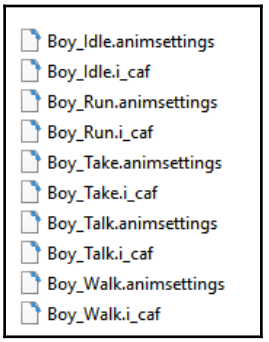
9. You will notice that when you click on any of the animations in the **Animations** section of the **Assets** pane, you receive an error in the Viewport. Click on the **Boy_Talk** animation to view the error:



10. To fix this error, we must specify a **Skeleton Alias** for the animation. We accomplish this by selecting the down arrow button to the right of **Skeleton Alias** field in the **Properties** pane and clicking on our farmer boy skeleton.
11. Next, click on the **Import** button:



12. Repeat steps *10–12* for each remaining animation. To ensure you accomplished this completely, click on each animation and make sure the error is no longer evident in the Viewport.
13. Use top menus of **Geppetto** to select **File | Save All**.
14. Right-click on any of the animations listed in the **Animations** section of the **Assets** pane and select **Show in Explorer**. You will see, as illustrated in the following screenshot, that Geppetto created the animation settings (`*.animsettings`) files. These files contain the compression settings for the animations:



Putting it all together

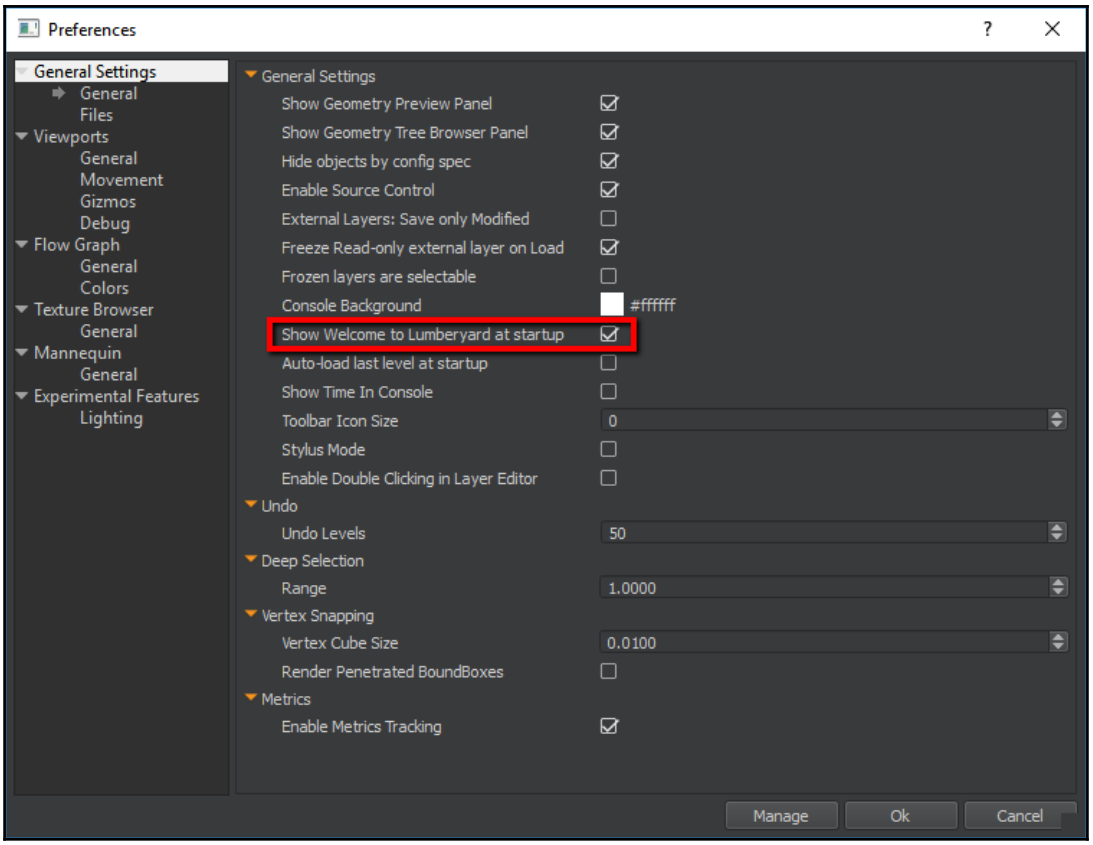
So far we have explored animation functionality with Mannequin and Geppetto. We understand that a third-party content creation system such as Autodesk Maya or Autodesk 3DS Max is required to create the animation files for import into our game levels. Next, let's take a look at a finished project to see how this all works together.

We will look at two types of animations: one that is initiated by user input and one that runs automatically.

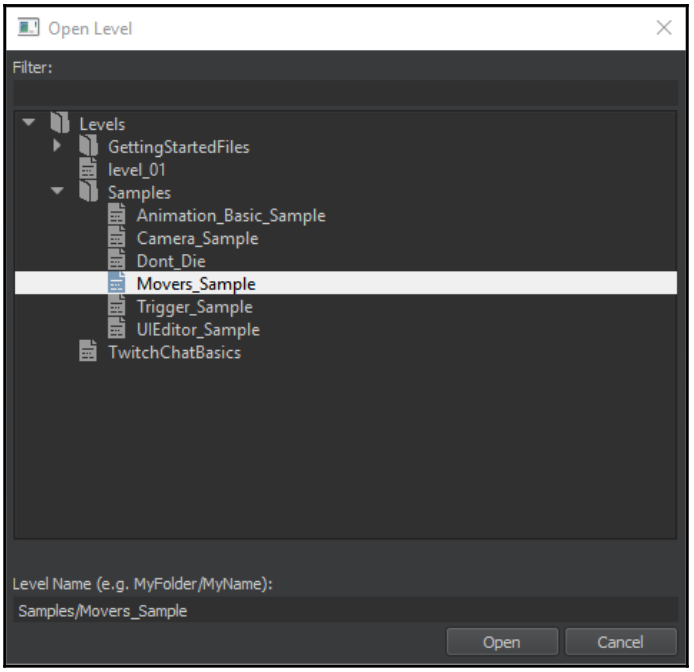
Animation triggered by user input

Follow these steps:

1. Launch the Lumberyard Editor.
2. On the **Welcome to Lumberyard Editor** dialog window, click the **Open level** button. If you previously checked the **Don't show this dialog on startup** checkbox in the **Welcome to Lumberyard Editor** dialog window, the window will not be present. You can toggle the window back on by selecting **File** | **Global Preferences** | **Editor Settings** and checking the **Show Welcome to Lumberyard at startup** checkbox. You can also select **File** | **Open** from the pull-down menu to launch the **Open level** dialog window:


Preferences
General Settings
General
Files
Viewports
General
Movement
Gizmos
Debug
Flow Graph
General
Colors
Texture Browser
General
Mannequin
General
Experimental Features
Lighting
General Settings
Show Geometry Preview Panel
Show Geometry Tree Browser Panel
Hide objects by config spec
Enable Source Control
External Layers: Save only Modified
Freeze Read-only external layer on Load
Frozen layers are selectable
Console Background
#ffffff
Show Welcome to Lumberyard at startup
Auto-load last level at startup
Show Time In Console
Toolbar Icon Size
0
Stylus Mode
Enable Double Clicking in Layer Editor
Undo
Undo Levels
50
Deep Selection
Range
1.0000
Vertex Snapping
Vertex Cube Size
0.0100
Render Penetrated BoundBoxes
Metrics
Enable Metrics Tracking
Manage
Ok
Cancel

3. Navigate to **Levels** | **Samples** | **Movers_Sample** and click the **Open** button:



4. Once the level loads, switch into Game Mode by pressing the *Ctrl* + *G* keys or selecting **Game** | **Switch to Game** from the pull-down menu. In Game Mode, you can control the player character with the mouse and *WASD* keys. You'll also notice that the space bar initiates a jump animation. Let's look at where the instructions are in Lumberyard that tell our character to jump when the Spacebar is pressed.
5. Depress your *Esc* key to exit Game Mode.
6. Select the character controller robot. You can zoom out so you can see the character controller in the Viewport.

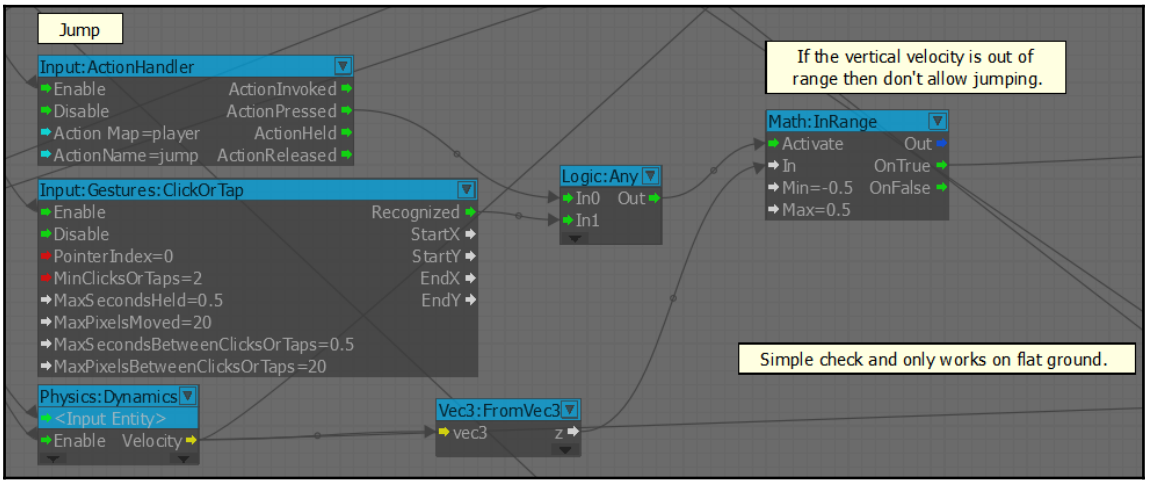
It can sometimes be difficult to select a specific object in a level that is full of objects. You can select an object from a list by using the **Edit** | **Select** | **Objects** from the pull-down menu.

7. You can right-click the **FG** icon that is part of the character controller to access the Flow Graphs. An easier way to access Flow Graphs is to select **View** | **Open View Pane** | **Flow Graph** from the pull-down menus.

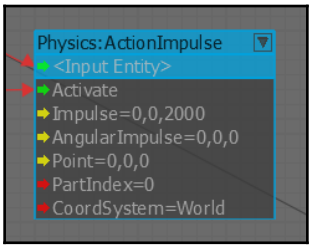
Flow Graphs are a visual scripting tool used in Lumberyard. **Flow Graphs** and the Flow Graph UI will be covered in `Chapter 6`, *Creating Gameplay*.

8. In the **Graphs** pane (lower left) of the **Flow Graph** dialog window, navigate to **Global** | **Modules** | **Character_Controller_Robot**.
9. In the **Canvas View** (the large middle area of the interface), zoom in so that you can read the **Jump graph**.

You can zoom in and out using your mouse scroll wheel.



After reading `Chapter 6`, *Creating Gameplay*, you will have a greater understanding of **Flow Graphs**. For now, you can see a series of inputs and outputs with a flow between component boxes. There is some validity checking and, if everything is okay, the **OnTrue** output is routed to the **Activate** component of the **Physics:ActionImpulse** component:



The **Impulse=0,0,2000** line in the **Physics:ActionImpulse** component indicates that, when activated, the player controller will move to the *XYZ* coordinates of **0,0,2000**. This is a relative location, so the *XY* coordinates of **0,0** indicate the current location of the character. The **2000** value is for the *Z* coordinate, which is the elevation. In the final steps, we will test the current **Flow Graph**, edit the **Impulse** value, and then test our changes.

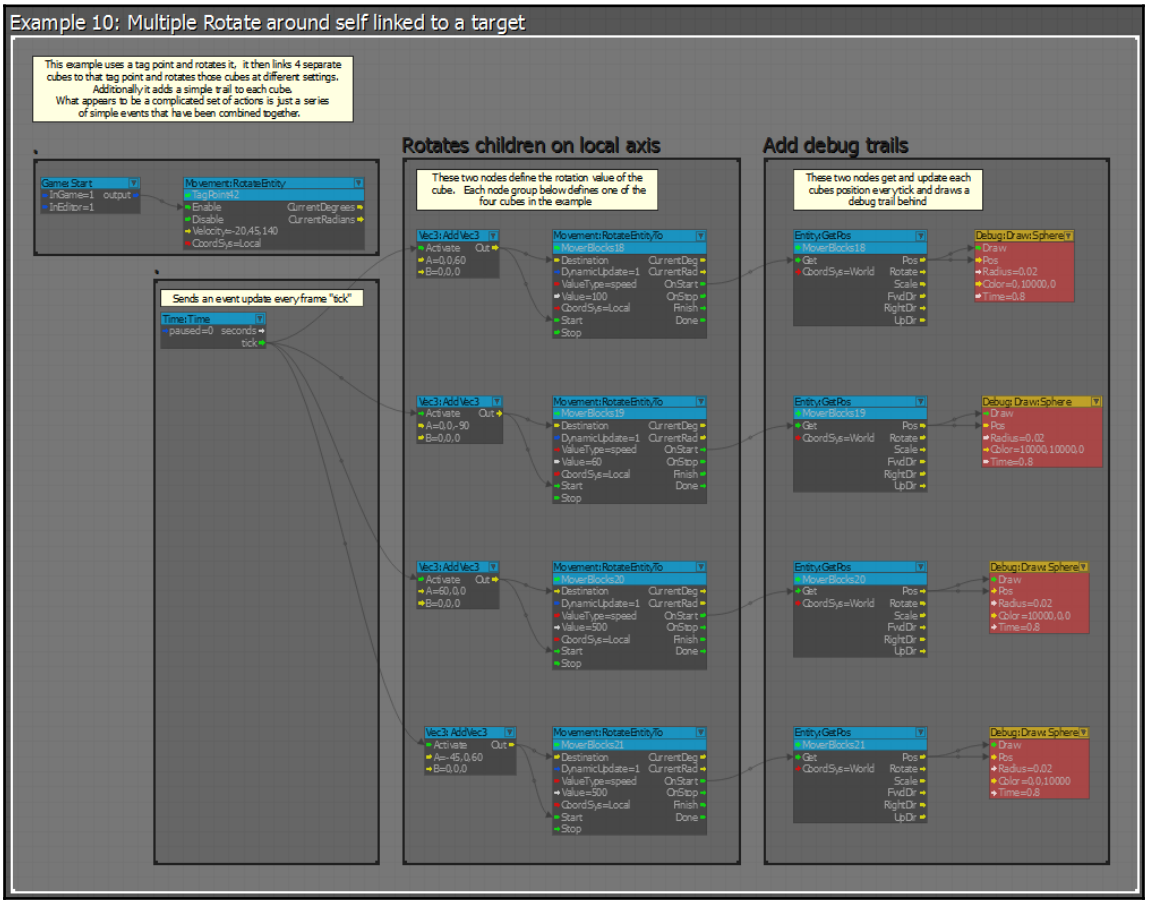
10. Switch to Game Mode by clicking on Lumberyard's main window. Use the *Ctrl* + *G* keyboard combination or **Game** | **Switch to Game** pull-down menu to test the game. Take notice of how high the character (robot) elevates when the spacebar is pressed.
11. Depress your *Esc* key to exit Game Mode.
12. In the **Flow Graph** window, select the **Physics:ActionImpulse** component box.
13. In the **Properties** pane, edit the **Inputs** | **Impulse** | **z** value, changing from `2000` to `5000`.
14. Switch back to game mode by clicking on Lumberyard's main window. Use the *Ctrl* + *G* keyboard combination or **Game** | **Switch to Game** pull-down menu to test the game. You will now see that the robot character elevates much higher.
15. Depress your *Esc* key to exit Game Mode.
16. In the **Flow Graph** window, select the **Physics:ActionImpulse** component box.
17. In the **Properties** pane, change the **Inputs** | **Impulse** | **z** value back to `2000`.

Automatic animations

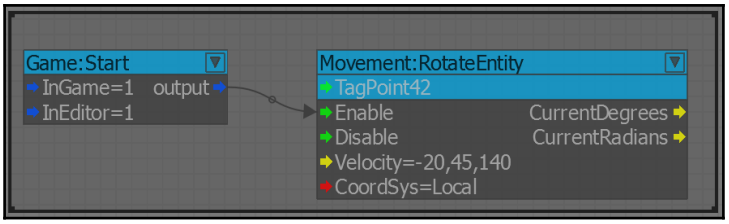
The second type of animations are those that run without the need for direct control by game players. Follow these steps to examine an automatic animation:

1. Launch the **Lumberyard Editor**.
2. On the **Welcome to Lumberyard Editor** dialog window, click the **Open level** button. If you previously checked the **Don't show this dialog on startup** checkbox in the **Welcome to Lumberyard Editor** dialog window, the window will not be present. You can toggle the window back on by selecting **File** | **Global Preferences** | **Editor Settings** and checking the **Show Welcome to Lumberyard at startup** checkbox. You can also select **File** | **Open** from the pull-down menu to launch the **Open level** dialog window.
3. Navigate to **Levels** | **Samples** | **Movers_Sample** and click the **Open** button.
4. Once the level loads, switch into game mode by selecting the *Ctrl* + *G* keys or selecting **Game** | **Switch to Game** from the pull-down menu.
5. Control the character controller, the robot, using your mouse and the *WASD* keys. View the multi-colored cubes. Notice that each of them have animations that continually run.
6. Depress your *Esc* key to exit game mode.
7. Select **View** | **Open View Pane** | **Flow Graph** from the pull-down menus to access **Flow Graphs**.

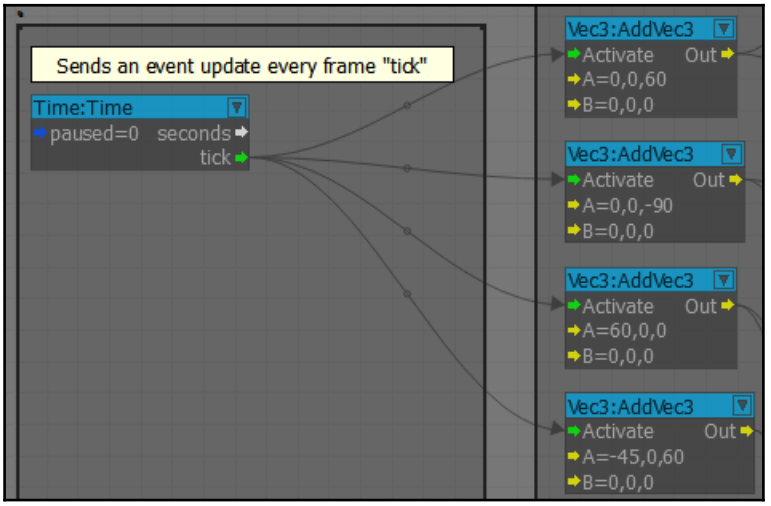
8. In the **Graphs** pane (lower left) of the **Flow Graph** dialog window, navigate to **Level | Entities | Example 10**. This displays the Flow Graphs in the **Canvas View**:



9. In the **Canvas View**, zoom in so that you can read the graph in the top-left section. You'll notice that the **RotateEntity** movement is enabled when the game starts:



10. The **Flow Graph** also contains an update event timer. As illustrated in the following screenshot, the output signal is sent with each frame tick. If our game is set to run at 30 frames per second, the output would communicate using the Activate input to the Vector nodes 30 times every second.



11. If desired, save your changes and exit Lumberyard.

Summary

In this chapter, we reviewed game animation concepts and discussed external content creation tools that are needed to create animations. In addition to revisiting Geppetto, you were introduced to Mannequin, Lumberyard's animation tool. The complexities of Mannequin were broached with suggestions to explore the game engine's documentation for advanced uses of the system. Mannequin's user interface was fully explored.

We also took a preview of Flow Graphs, Lumberyard's visual scripting system. We used Flow Graphs to look behind the green curtain and learn how animations are governed. Both user-initiated and automatic animations were covered. This preview of Flow Graphs will serve as a nice primer for the next chapter.

In the next chapter, we will start creating gameplay. This will include a look at game logic, AI, and game physics. We will look at pre-built examples and create gameplay of our own.

6
Creating Gameplay

In the previous chapter, we explored animations of characters and objects. We looked at Mannequin, Lumberyard's animation tool, and continued our use of *Geppetto*, Lumberyard's character tool. Animations help bring a game to life, but they have no value on their own. In order to make a game interactive, we need to create gameplay components. That will be the focus of this chapter.

After reading this chapter, you will:

- Understand the gameplay concept
- Have explored an existing sample game
- Have become familiar with the Flow Graph UI
- Be able to read Flow Graphs
- Have made changes to Flow Graphs to impact game AI
- Have edited Flow Graphs to change game physics

Understanding gameplay

Gameplay can be seen as interactions between a player and the game and between players. When a set of rules for a game is adopted and the game starts, gameplay can take place.

Gameplay is often written as *Gameplay* and *Game Play*. Neither form is incorrect, nor is one more correct than the other. This book adopts the *Gameplay* usage.

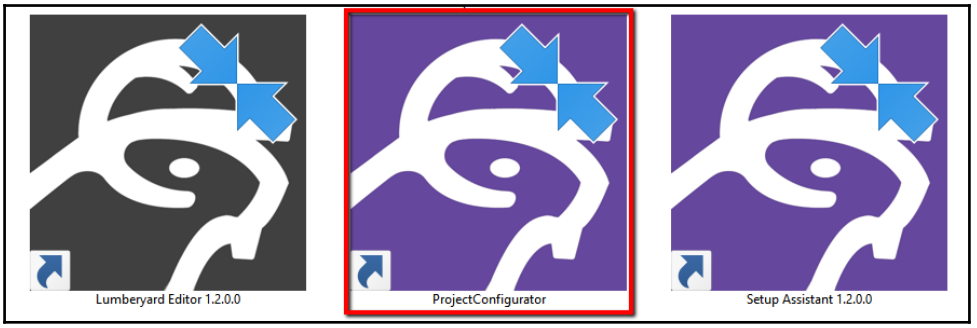
Examples of gameplay include when you hit the Spacebar, your character jumps; when you come within range of a gun turret, the turret shoots at you; and so on. In the broader sense, gameplay is what permits the game to be played and the parameters in which it takes place.

Lumberyard is a fully-fledged AAA game engine that permits great affordances to introduce gameplay into your games. Later in this chapter, you will gain great insight into how a gameplay is created and managed.

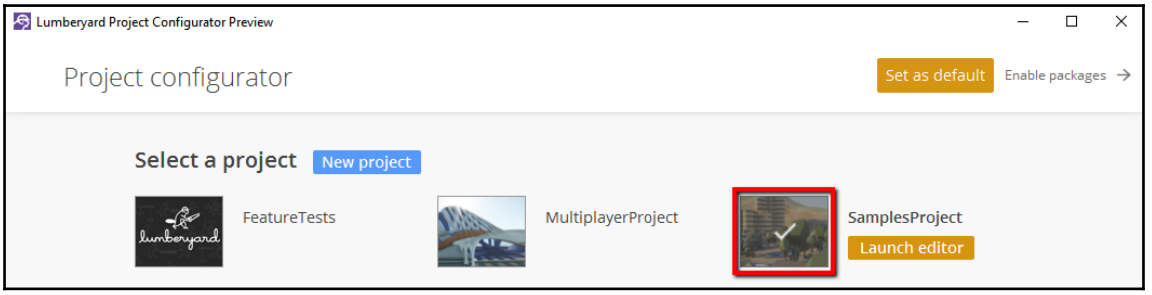
Getting started

In order to learn how to create a gameplay using Lumberyard, we will start by opening a complete game. Amazon has a few samples that can be downloaded from the Lumberyard downloads page as well as some that are part of the game engine download. The following steps will configure the Lumberyard Editor to use the **SamplesProject** already installed with Lumberyard:

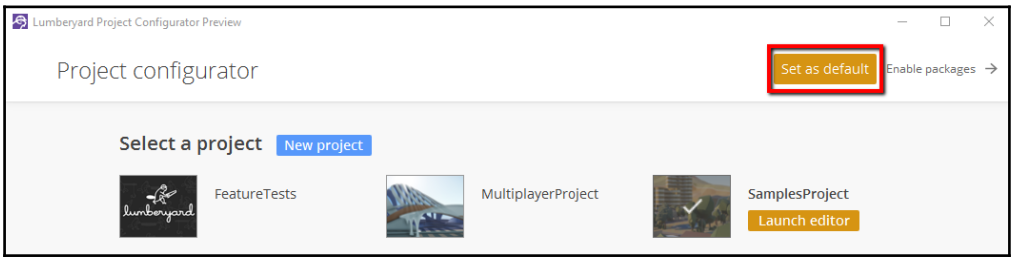
1. Launch the Lumberyard **Project Configurator**. If you do not have the shortcut icon on your desktop, you will be able to find the app in the `dev\Bin64` folder of your Lumberyard install.



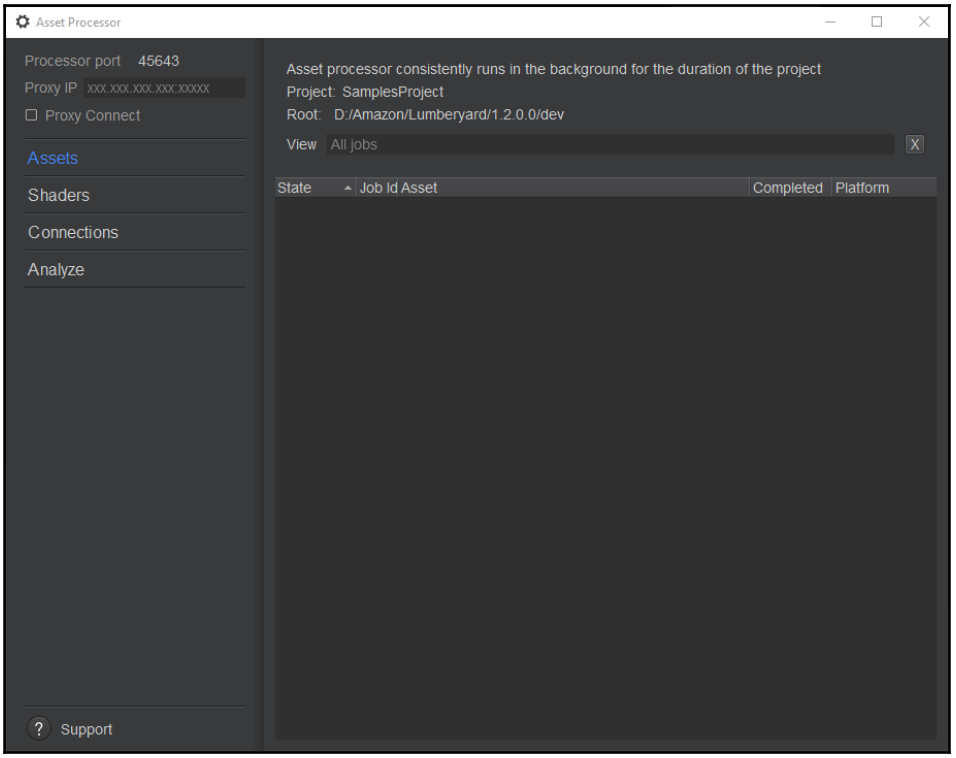
2. In the **Project Configurator**, select the **SamplesProject** icon.



In the **Project Configurator**, click the **Set as default** button. This will make our newly selected game the default. This ensures we have ready access to the game's assets from within Lumberyard.

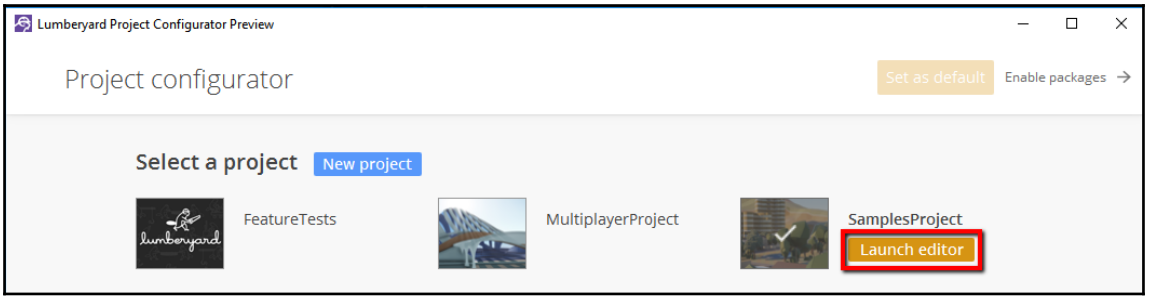


3. The **Asset** Processor will immediately start processing the game's assets. This can take a few minutes.



You do not need to close the **Asset Processor** window. The processor will run in the background whenever you are editing a project in Lumberyard.

4. Our last step is to load the game in Lumberyard. Depending on your Lumberyard version, you will either click the **Launch editor** button in the Project Configurator or launch the **Lumberyard Editor** from a desktop icon.



5. In the **Welcome to Lumberyard Editor** dialog window, click the **Open level** button. Alternatively, you can select **File** | **Open** from the drop-down menu.
6. Select **Levels** | **Samples** | **Camera_Sample** and click the **Open** button. This will load a game with one level that we will explore and modify throughout the remainder of this chapter.

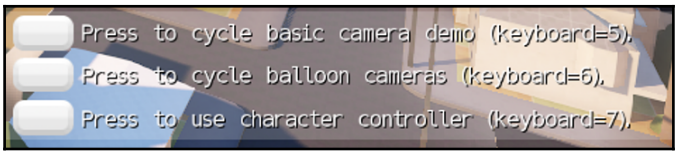
Now that you have the game sample loaded, we will look at what gameplay already exists, make modifications, and create our own gameplay.

Exploring the Camera_Sample game

Now that the **Camera_Sample** game is loaded, switch to Game Mode and experiment with the game. Try to identify gameplay components.

You can switch to Game Mode with the *Ctrl* + *G* keyboard combination or by selecting **Game** | **Switch to Game** from the pull-down menu system. The *Esc* key will return you to the **Lumberyard Editor**.

The first thing you will notice is that a **Heads Up Display** (**HUD**) presents you with three game modes.

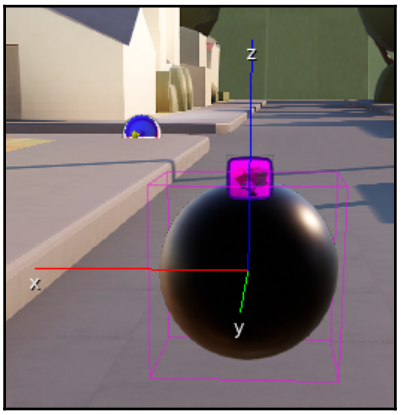


Basic camera demo

Depressing the 5 key in Game Mode starts a demo in first-person mode with the character travelling around a street block in the game environment. Each time you press the 5 key, the game perspective changes; see the following table for details:

Iteration Number.	**Game Perspective**
1	First-person point of view
2	Third-person point of view
3	Third-person shoulder point of view
4	Top down
5	Side scroller
6	Parented tracking + dynamic zoom
7	Tracking + dynamic zoom
8	Chase
9	Top down player
10	Simple character controller

This demo is looped. As you can see from the following screenshot, the character used in this mode is a sphere with a camera centered on top of it:



Since this game mode is presented in the first-person, you do not see the sphere during gameplay. In the *Flow Graph* section of this chapter, we will examine how this demo's gameplay was created.

Balloon camera demo

This game mode is accessible by pressing the 6 key when the game launches. It uses a camera that tracks the group of balloons. This is an excellent example of what can be done to create cut-scenes, gameplay videos, and even creating a gameplay where an orb or missile tracks targets. We will explore how this demo was created in the *Flow Graph UI* section of this chapter.

Character Controller mode

This game mode is accessible by pressing the *7* key when the game launches. It is interactive and not a demo. Third-person perspective is used, which allows us to see our robot character as we control it. The *WASD* keys and mouse are used to control the character. The Spacebar provides temporary elevation, or a jump to the robot. This allows the robot to jump onto the sidewalk.

In this mode, you can control the robot to explore the game's environment. It consists of a 15-block residential area with numerous homes, apartment buildings, trees, street lights, power poles, and very clean streets. There is an empty swimming pool structure in the game that you can traverse, a basketball court, and a tennis court.

There are also a few things happening that take place seemingly regardless of any user actions:

- Cylindrical object that travels around the block
- Stationary cylindrical object
- Orb in swimming pool that leaves a trail trace

We'll explore each of these after we take a tour of the Flow Graph UI.

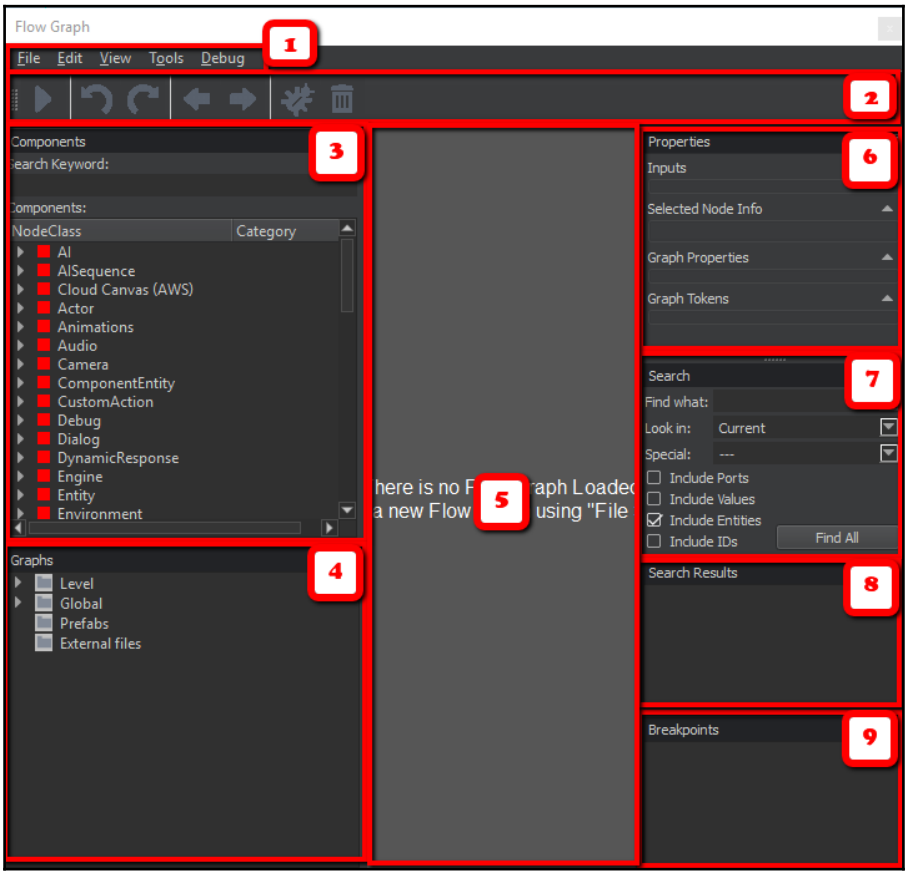
Understanding the Flow Graph system and UI

Flow Graph is a visual scripting system used to create gameplay in Lumberyard. It is, of course, much more complex than that and is used to create and edit in-game events, AI, game logic, and other aspects of games you create. We'll explore the Flow Graph UI and then review graphs for the game modes detailed in the previous section.

Flow Graph is the system that is used to create flow graphs, are organized in modules. These graphs can be viewed as gameplay blueprints. Graphs comprise of nodes with inputs and outputs. These graphs, or blueprints, determine a game's constraints, rules, and gameplay.

Flow Graph UI

Flow Graph is accessible through the **View** | **Open View Pane** | **Flow Graph** pull-down menu. There are 10 unique interface components. The following screenshot assigns numbers to each component and this is followed by a detailed explanation:



The following table provides component names for each **Flow Graph** interface area, as depicted in the previous screenshot:

Reference	Component Name
1	Pull-down menu system
2	Hot keys
3	**Components** pane

4	**Graphs** pane
5	Viewport
6	**Properties** pane
7	**Search** Pane
8	**Search Results** pane
9	**Breakpoints** pane
10	**Multiplayer** pane (not shown)

The pull-down menu system

The pull-down menu system (number **1** in the previous screenshot) consists of four top level options: **File**, **Edit**, **View**, **Tools**, and **Debug**.

The File menu

The **File** menu lets you create a new graph. When a new graph is created, it will be placed in the hierarchical list (refer to the *Graphs pane* section). New graphs are given the name `Default` and can be changed:

- The **New Flow Graph Module** option allows you to create **Global** and **Level** modules. Selecting this option requires you to select **Global** or **Level** and then a location where your module will be saved. Flow Graph modules are saved as XML (`*.xml`) files. **Global** modules can be used throughout your project, while **Level** modules are specific to a level.
- The **New MaterialFX Graph** option allows you to create material effect graphs. They are saved as XML files.
- The **Open** option gives you the opportunity to open an externally saved graph.
- The **Save** option lets you save your graphs.
- The **Save As** option lets you save your graphs with a different name and location.
- The **Import** option allows you to import graphs from other levels for use in the level you are currently working on.
- The **Export Selection** option allows you to export a copy of a graph as a module. You can export graph modules from one project and import it in to another.

The Edit menu

The **Edit** menu has a standard set of editing functionalities. There are sets of undo/redo and group/ungroup options. There are also the typical cut, copy, paste, paste with links, and delete functions. In addition, there is a find option to help you find specific entities in your graphs. The find function uses the `Search` pane (component **7**). Additional functionality exists for zooming in, zooming out, and refreshing the view.

The View menu

The **View** menu consists of a series of interface component toggles. You can toggle on/off the following screen areas:

- **Components**
- **Graphs**
- **Properties**
- **Breakpoints**
- **Search**
- **Search Results**
- **Multiplayer**

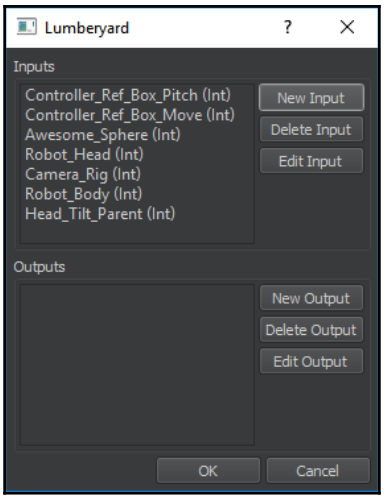
You can also indicate, by using the second **Components** item, which types of entities you want included: release, advanced, debug, or obsolete.

The Tools menu

The **Tools** menu offers two functions: **Edit Graph Tokens** and **Edit Module**. A token is a variable that is used within a single graph. An example might be a health meter token that a graph changes based on the player sustaining damage or obtaining health items.

If you are familiar with programming, graph tokens are essentially local variables.

The second function, **Edit Module**, provides a means to edit the currently selected module. If you do not have a module selected, this feature will have no effect. Once a module is selected, and the Edit Module option is invoked, the **Module Editor** window is presented. Here you have access to create, edit, and delete inputs and outputs from the selected module.



The Debug menu

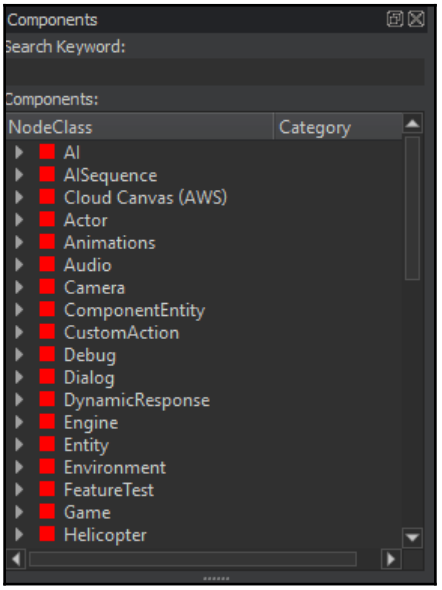
The final top-level menu option is **Debug**. Using this menu, you can enable debugging, erase debugging information, and indicate whether or not you want to omit any specific flow graph types from the debugging information. These types are **AI Action**, **Module**, **Custom Action**, and **Material Effects** (**MaterialFX**).

Hot keys

Hot keys are referenced as component **2** in the previous **Flow Graph** UI screenshot. There are seven hot keys located directly under the pull-down menu system. The buttons, ordered left to right (see the following screenshot), are **Play**, **Undo**, **Redo**, **Back**, **Forward**, **Enable Debugging**, and **Erase Debugging Information**.

Components pane

The **Components** pane (referenced as component **3** in the preceding **Flow Graph** UI screenshot) displays all available flow graph nodes. The nodes are organized by **NodeClass**.

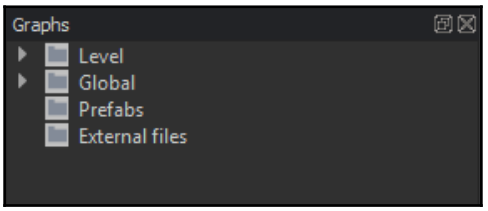


These nodes are template nodes that you can use in constructing flow graphs. To the left of each **NodeClass** is an arrowhead icon that allows you to view and hide the individual nodes in each class. Once expanded, you will see that each node also has a release, advanced, debug, or obsolete category.

Exploring each node will give you great insights into what is possible when creating a gameplay with Flow Graph.

Graphs pane

The **Graphs** pane (referenced as component **4** in the previous **Flow Graph** UI screenshot) is a hierarchical list of your graphs. When you right-click a graph, you can create a new global module, delete the module, and enable or disable breakpoints.

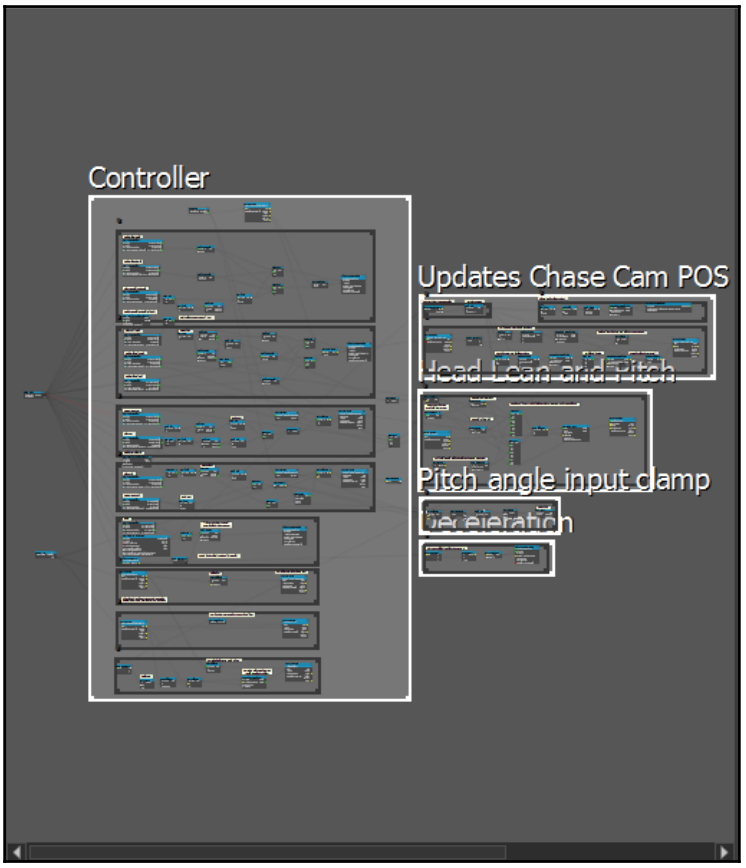


When you select **File** | **New** from the pull-down menu, a new graph will appear under **External Files** in the **Graphs** pane. Right-clicking that graph will let you disable the graph, rename it, delete it, change the folder, and enable or disable breakpoints.

Viewport

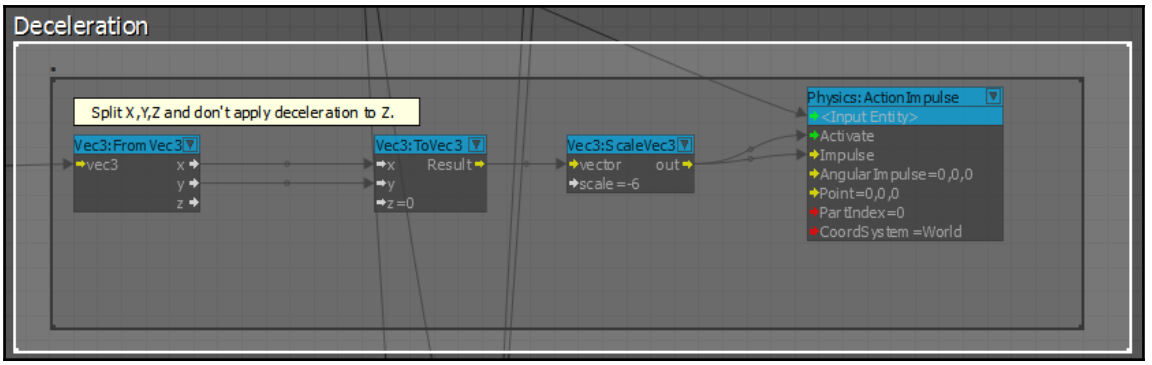
The Viewport (referenced as component **5** in the previous **Flow Graph** UI screenshot) is the main editing window for the **Flow Graph** system. When you view a graph in the Viewport, it might seem difficult to see the details. This is because graphs can be very complex and too large to view in a single view. You can zoom in and out of the graph, by using your mouse scroll wheel. You can also move the graph within the Viewport by holding down the right mouse button and then dragging.

The following screenshot shows a zoomed out image of the Viewport:



You will notice that at the bottom of the Viewport is a horizontal scrollbar. That indicates that there is more graph than is currently visible in the Viewport. When the vertical scrollbar is visible, there is additional content above and/or below the current view. You can manipulate both scrollbars with your mouse.

The next screenshot displays a zoomed in view of the **Deceleration** section of the graph. We will look at individual nodes as well as inputs and outputs later in this chapter.

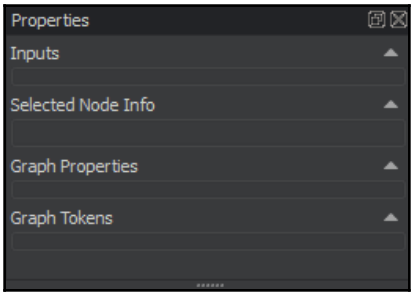


To select a node, you simply click on it. When you right-click a node, you are presented with additional functionality. You can rename the node, manage inputs and outputs, and have access to standard editing functionality (cut, copy, paste, paste with links, and delete).

An additional option is to display or suppress the **Spline Arrows**. The lines that connect nodes are, by default, shown as Spline Arrows. You can uncheck this option to view the connecting lines as straight lines.

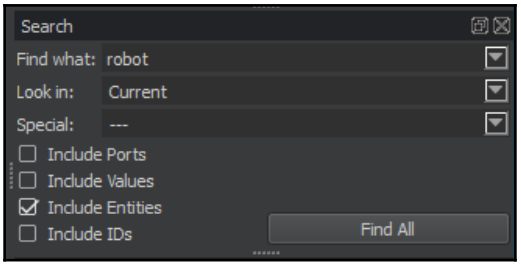
Properties pane

The **Properties** pane (referenced as component **6** in the previous **Flow Graph** UI screenshot) displays contextual properties based on the selected node. You can set and edit values in this pane.



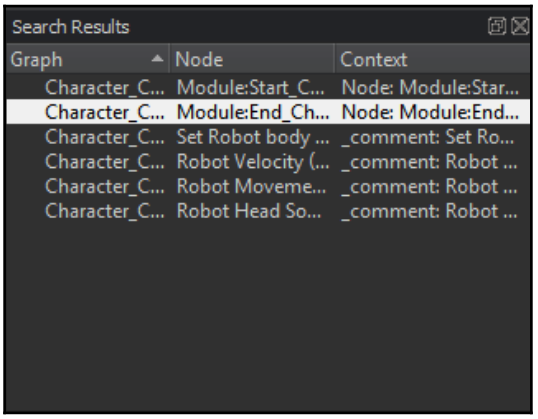
Search pane

The **Search** pane (referenced as component **7** in the previous **Flow Graph** UI screenshot) is the interface you will use to search components of your graphs. Games can be very complex, which means you will have a lot of graphs and many of them will be very complex. The search functionality is a handy tool.



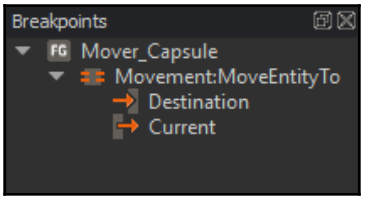
Search Results pane

The results of searches are populated in the **Search Results** pane (referenced as component **8** in the previous **Flow Graph** UI screenshot). The results will be displayed in three columns: **Graph**, **Node**, and **Context**. Double-clicking a line will focus the Viewport so that the selected component is center-stage.



Breakpoints pane

If you add **Breakpoints** to your graph, they will be displayed in the **Breakpoints** pane (referenced as component **9** in the previous **Flow Graph** UI screenshot). To add a breakpoint to an input or output of a node, simply right-click the input or output point and select **Add Breakpoint**.



Breakpoints allow you to introduce points in the game where the gameplay will pause. This is used for testing and debugging your game.

Multiplayer pane

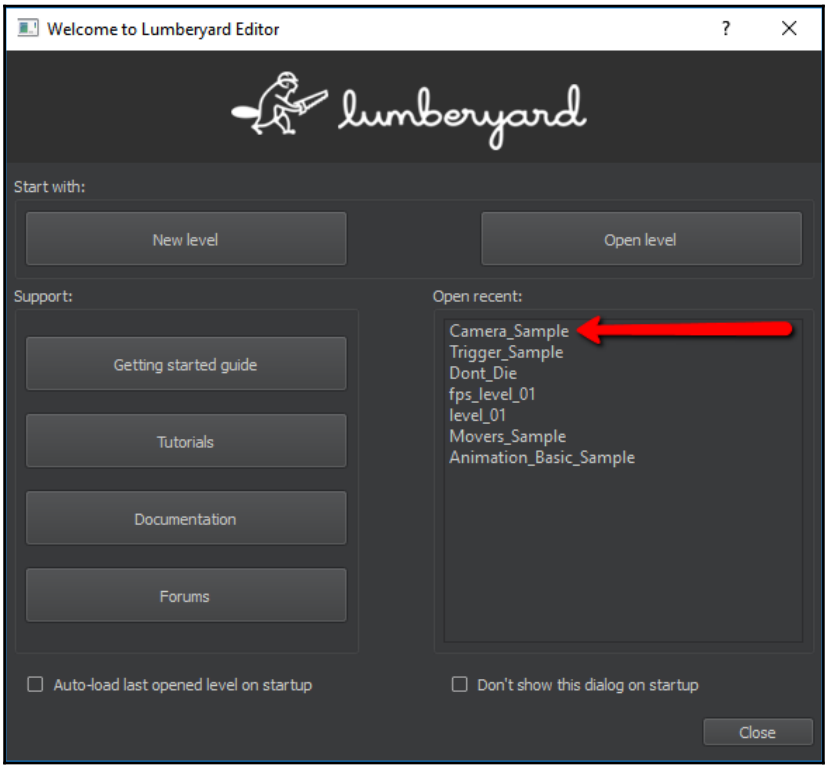
The **Multiplayer** pane is only available if you have a multiplayer game. This pane was not presented in the previous **Flow Graph** UI screenshot, and was included in the Flow Graph UI Component table to signify that it is available based on contextual relevance.

Game example review

In this section, we will accomplish two tasks using the Camera Sample game we previously explored in this chapter. First, we will examine graphs to gain an understanding of how they work. Next, we will make modifications to graphs to change the gameplay. Then, in the next section, we will create our own graph for additional gameplay.

To get started, let's load the Camera Sample game:

1. Launch Lumberyard.
2. Since we previously opened the Camera Sample game level, the level will be listed in the **Welcome to Lumberyard Editor** dialog window's **Open recent** listbox. Click the **Camera_Sample** level in the **Open recent** listbox.



3. Now that you have the level loaded, let's open **Flow Graph**. Select **View** | **Open View Pane** | **Flow Graph** from the pull-down menu.

Chasing the rabbit

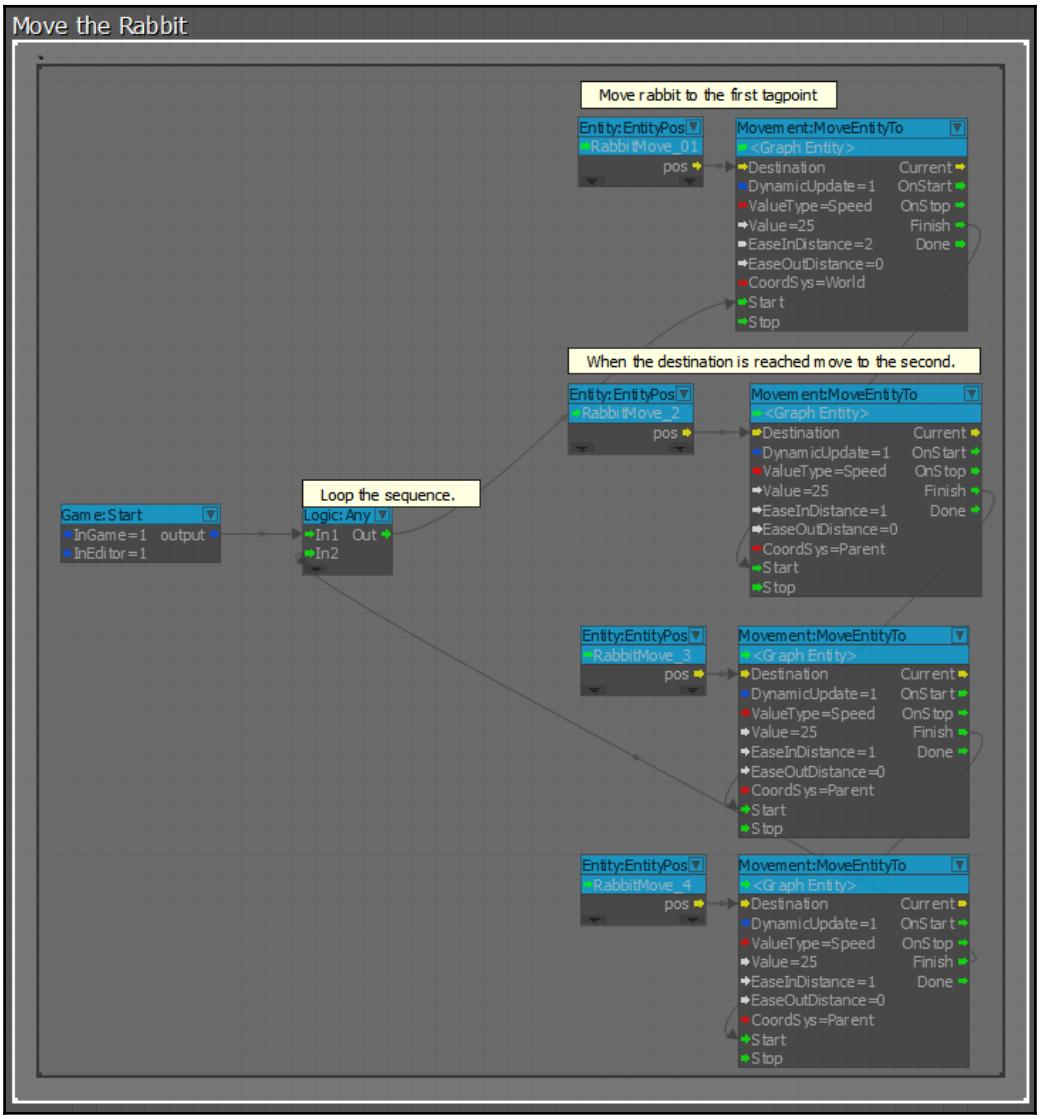
The sample game includes a white cylinder that travels in a loop around the city block containing the swimming pool. It accomplishes this with gameplay established by two graphs: **Rabbit** and **Mover_Capsule**.

In the **Lumberyard Editor**, you can see the presence of a rabbit object and four target points. These are labeled in the following screenshot:

We will now review two graphs and use the previous screenshot as a reference.

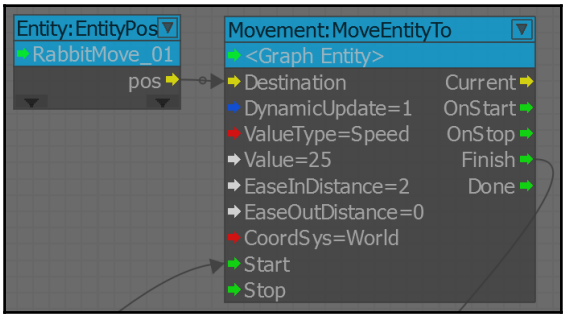
Rabbit graph

The **Rabbit** graph is accessible in Flow Graph by selecting **Level** | **Entities** | **Physics** | **Rabbit** in the **Graphs** pane. This graph has several nodes, so you might need to zoom in to see individual components.

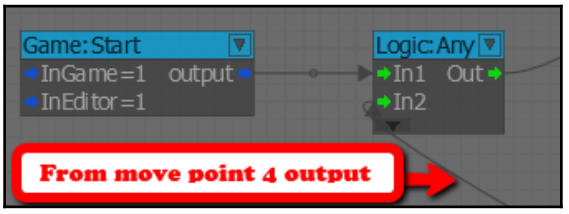


Locate the **Game:Start** node. It has two inputs on the left: **InGame** and **InEditor**. This enables this graph when the game starts regardless of whether it is being played in game mode or in the editor. Output from this node maps to the input of the **Logic:Any** node. This is the loop node. You'll note that there is a second input that comes from the graph when the rabbit has moved to the final move point.

The logic node outputs to the **Start** input of the first **Movement:MoveEntityTo** node. As you can see in the following graph segment, the first **Movement:MoveEntityTo** node also takes the first move point's vector position as the **Destination** input. The **DynamicUpdate** Boolean value is **1**, which causes the entity (the **Rabbit**) to move to the destination. Hover over additional values to see details.



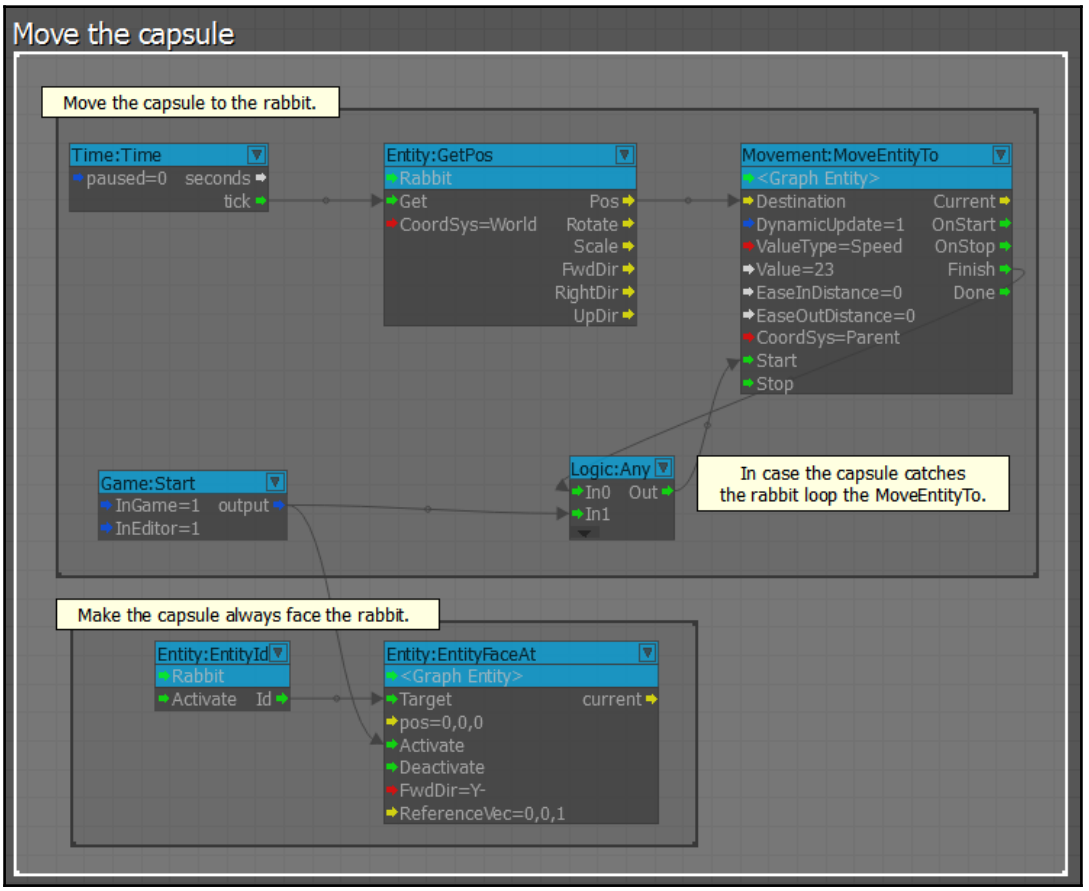
When the rabbit has moved to the destination (move point *1*), the move is finished. The **Finish** output feeds to the input of the second **Movement:MoveEntityTo** node. This is repeated for the third and fourth nodes. Once the rabbit has been moved to all four sequential move points, the graph processing ends. The final step in the graph is to connect the output from Finish on the fourth **Movement:MoveEntityTo** node to the input of the loop node.



The second graph involved in the cylinder's movement is the **Mover_Capsule** graph. We'll examine that graph next.

Mover_Capsule Graph

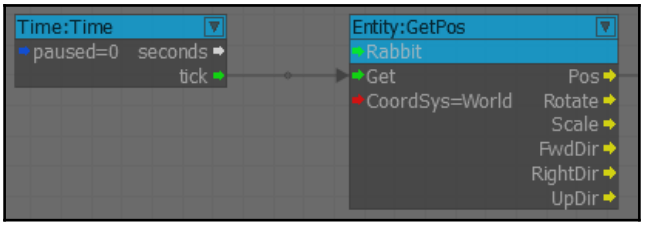
The **Mover_Capsule** graph is accessible in **Flow Graph** by selecting **Level** | **Entities** | **Physics** | **Mover_Capsule** in the **Graphs** pane.



As you can see in the **Mover_Capsule** graph, c c inputs to the **Game:Start** node activates the graph and has two outputs. The first output activates the **Entity:EntityFaceAt** node. This node forces the capsule to always face the rabbit. This is the natural way it would be if the capsule was truly chasing a rabbit. The following screenshot shows the capsule facing the rabbit:

The second output from the **Game:Start** node passes into an input on the **Logic:Any** node and then outputs to activate the **Movement:MoveEntityTo** node. This node is of the same type as used with the **Rabbit** graph. For our **Mover_Capsule** graph, the **Movement:MoveEntityTo** node takes the destination point from the rabbit's current position.

As illustrated in the following screenshot, the **Entity:GetPos** node is processed every tick. It retrieves the rabbit's current position and outputs that to the **Movement:MoveEntityTo** node.



So, as the rabbit travels around the block, the capsule maintains the same position as the rabbit.

You'll remember from *Chapter 4*, *Creating 3D Characters*, that a tick is the length of one frame. So, if your game is 30 **Frames per Second** (**FPS**), a node set to process each tick will be processed 30 times per second.

Editing a Flow Graph

Now that you are armed with the ability to navigate and read graphs, let's edit existing graphs. First, we will edit the speed that the capsule travels to follow the rabbit:

1. Launch Lumberyard.
2. Open the **Camera_Sample** game.

3. Select **View** | **Open View Pane** | **Flow Graph** from the pull-down menu.
4. In Flow Graph, select **Level** | **Entities** | **Physics** | **Mover_Capsule** in the **Graphs** pane.
5. Select the **Movement:MoveEntityTo** node in the Viewport.
6. In the **Properties** pane, change the **Value** to `5`. This will drastically change the speed in which the capsule follows the rabbit. The rabbit's speed is set at `25`, so the capsule is likely to have problems keeping up.
7. Switch to Game Mode and watch the capsule's behavior. You will see that the capsule moves much more slowly than normal and does not come close to navigating the block. In fact, the capsule now travels in a much smaller area as it is always headed towards the rabbit's current position.
8. Change the **Value** to `1` and re-test the game. Now, the capsule's area of movement is much smaller.
9. Change the **Value** to `45` and test the game again. You can see how easy it can be to make changes to graphs.
10. Reset the **Value** to `23`.

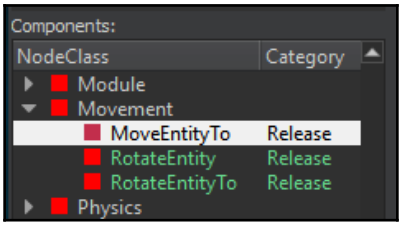
Next, we will make changes to the game using the **Lumberyard Editor** and the **Rabbit** graph to change the path in which the rabbit travels and, therefore, where the capsule travels:

1. In the **Lumberyard Editor**, type `Rabbit` in the search box located above the **Perspective Viewport** and press the *Enter* key. This will change your view so that only filtered results will display in the Perspective Viewport.

2. Select one of the rabbit move points.
3. Using the pull-down menu, select **Edit** | **Clone**. This creates a copy of the move point and automatically increments the number.
4. Select **RabbitMove_5**. This is the cloned item you created in the previous step. Relocate it across the street. You can drag the move point into position or manually enter `975`, `1064`, `50` as the coordinates.

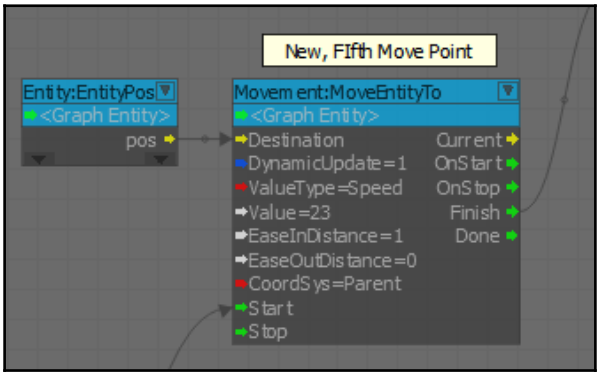
If you do not see move handles on your cloned object, click the **Select and Move** icon on the **Quick Access Bar**. Once selected, you can click any of the *X*, *Y*, or *Z* handles to move it along the associated vector. You can also enter the data in the *X*, *Y*, and *Z* fields at the bottom of the interface.

5. Clear the filter by clicking the circled-**X** in the search bar.
6. Next, we will edit the **Rabbit** graph to include the fifth move point.
7. In **Flow Graph**, select **Level** | **Entities** | **Physics** | **Rabbit** in the **Graphs** pane.
8. In the **Components** pane of **Flow Graph**, select the **Movement** | **MoveEntityTo** node.



9. Drag the **Movement:MoveEntityTo** node to an open space in the Viewport.
10. Next, we will add an input to the Destination. In the **Components** pane of **Flow Graph**, select the **Entity** | **EntityPos** node.
11. Drag the **Entity:EntityPos** node to an area to the left of the **Movement:MoveEntityTo** node you previously created.
12. Right-click the **Choose Entity** line of the **Entity:EntityPos** node. Select the **Assign Selected Entity** option. This assigns the **RabbitMove_5** move point to this node.
13. Next, we will connect the output of the **Entity:EntityPos** node to the Destination input of the **Movement:MoveEntityTo** node. Click on the yellow arrow to the right of the **pos** element of the **Entity:EntityPos** node and drag it to the yellow arrow to the left of the **Designation** input of the **Movement:MoveEntityTo** node.
14. We will now make changes to the **Movement:MoveEntityTo** node you previously created. Right-click the **Choose Entity** line of the node and select the **Assign Graph Entity** option.
15. With the node selected, change the **Value** from 0 to `23` in the **Properties** pane.
16. Change the **EaseInDistance** to `1`.
17. Next, we will connect the fourth move point to the new (fifth) move point. Currently, the **Finish** output on the fourth move point is connected to input **2** of the loop node. Click the green arrow to the left of the **In2** input of the **Logic:Any** node and drag it to the **Start** input of the fifth move point.

18. The last step is to connect the **Finish** output of the fifth move point to the **In2** input of the **Logic:Any** node. When you are finished, your new nodes should resemble the following screenshot:



You can now put your level into Game Mode to test your results.

Creating a new Flow Graph

So far, we have explored and edited existing flow graphs. In this section, we will create a new graph so that you can experiment on your own without the danger of breaking any existing flow graphs:

1. Launch Lumberyard.
2. Open the **Camera_Sample** game.
3. Select **View** | **Open View Pane** | **Flow Graph** from the pull-down menu.
4. In **Flow Graph**, select **File** | **New** from the pull-down menu to create a new graph.
5. Right-click and select **Rename Graph**, then change the name to `Chapter6Testing`.

Our flow graph has been created and, although it is empty, it is ready to use. One of the best ways to learn how to create flow graphs is to practice. Take a simple gameplay and try creating a flow graph for it. You can learn a lot using this approach.

Summary

In this chapter, we explored the concept of gameplay and how it can be instantiated in Lumberyard games. We took an in-depth look at a sample game that ships with Lumberyard. Specifically, we reviewed the game's gameplay and examined the flow graphs that determined how the gameplay mechanics work.

We also thoroughly explored the Flow Graph UI. This visual scripting system is complex and can be used to create complete games within Lumberyard. You were left with the ability to read, edit, and create flow graphs.

In the next chapter, we will look at the requirements of creating a multiplayer game in Lumberyard. We will also explore *Amazon GameLift*.

7
Creating Multiplayer Gameplay

In the previous chapter, we explored the concept of gameplay and how gameplay is created in Lumberyard. Our exploration included an in-depth look at a sample game. We also became familiar with the Flow Graph system, Lumberyard's visual scripting system. We used Lumberyard's Flow Graph system to view, edit, and create gameplay.

In this chapter, we will examine the requirements for creating a multiplayer game in Lumberyard. We will also explore *Amazon GameLift*.

After reading this chapter, you will:

- Understand some of the major multiplayer gameplay considerations
- Understand the need for game servers
- Establish your business or personal **Amazon Web Services** (**AWS**) account
- Examine multiplayer game considerations
- Create a multiplayer game using GameLift

Multiplayer gameplay considerations

Multiplayer games are among the most difficult to develop. This is due to the complexities involved in networking and game servers, and frankly the number of things that can go wrong. You likely have played a lot of multiplayer games. Some of the common things that can go wrong include:

- Latency/lag
- Crashes/disconnects
- Slow loading

- Delayed spawning
- Players having different game versions
- Maps not available

So, you are familiar with some or all of these problems. There are also design considerations that must be designed and developed. Here are a few:

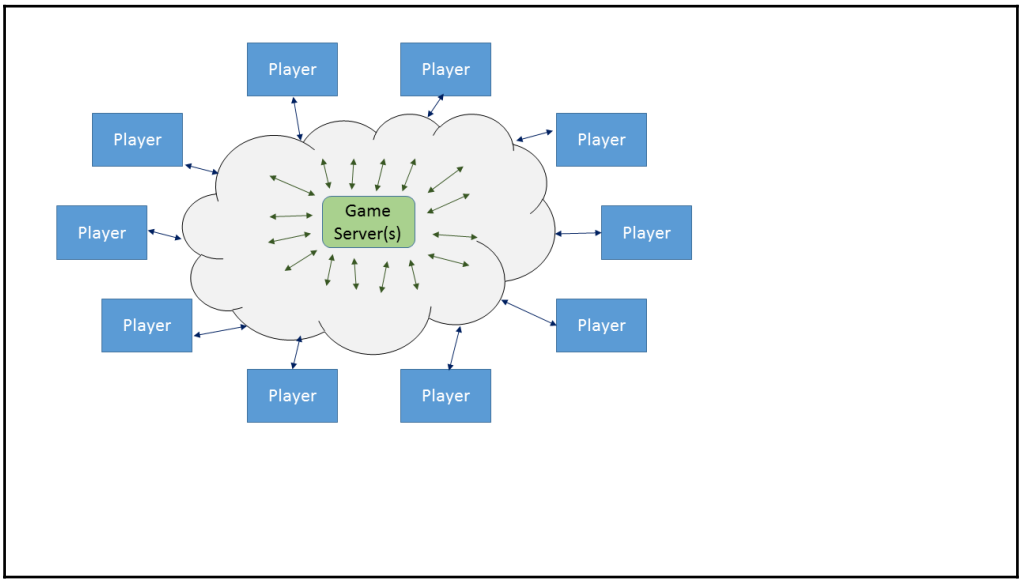
- Server-side programming (game servers)
- Rooms and lobbies
- Battle system balance
- Team and cooperative modes
- Creating battle arenas
- AI and **Non-player Characters** (**NPC**) behaviors

You'll create your game in Lumberyard and not worry too much about server-side programming, because Amazon GameLift will take care of this for you. GameLift works in concert with multiplayer games developed with Lumberyard.

The need for game servers

One of the greatest challenges to overcome in developing multiplayer games is the server requirements. Few of us have servers connected to the Internet that can be used to host multiplayer games. Those that do still have to deal with the issue of scalability. Let's say you have a server environment that can handle up to 20 simultaneous games, each with up to 12 players. All of a sudden, your game becomes popular, you now have several hundred game requests, and your server crashes. The ability to scale on the fly is one of the great benefits of using Lumberyard.

Here is an illustration of how multiple players connect to one another in multiplayer games. This is an over simplification, but adequately indicates the primary points of interest:



Each player has the game software running on their own computer. The game might run within a browser, or have been installed on a hard drive. Each player is connected to the Internet and is routed to the game's server or servers. The software on the game server controls how players connect to the game, what players are linked with others, and typical multiplayer game functions such as lobbies, rooms, chat, achievements, scores, respawning, and more.

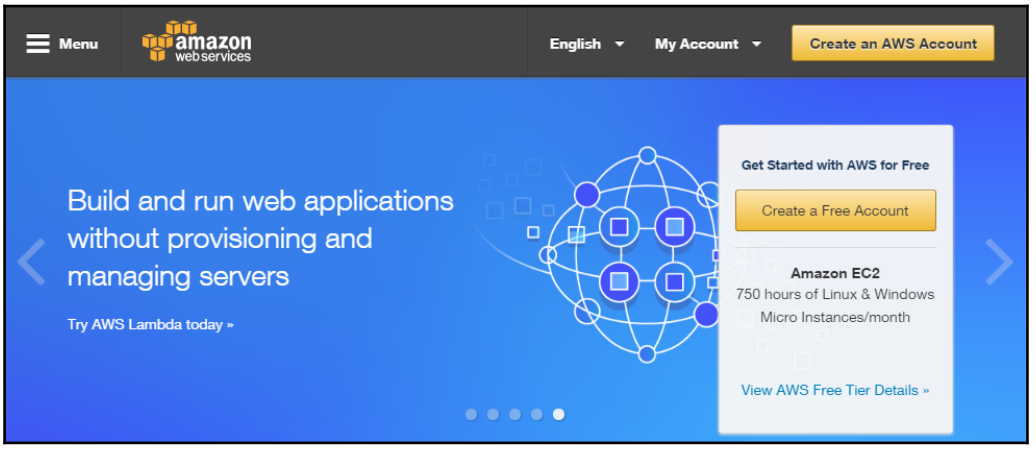
Lumberyard does a lot of the heavy lifting with regards to the server part of multiplayer games. Lumberyard integrates with GameLift, which is part of the AWS offering. In the next section, you will learn how to establish your AWS account and get started using GameLift.

Understanding AWS

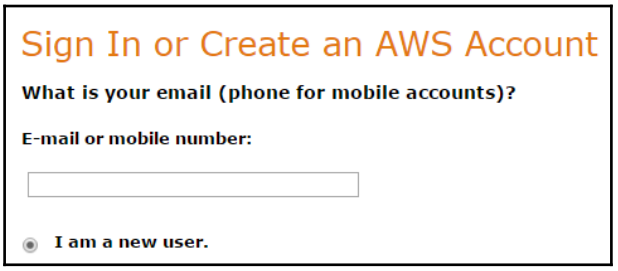
Amazon Web Services, also referred to as AWS, is a cloud platform that offers a multitude of scalable services. You can learn about the various services offered by AWS at `aws.amazon.com`. We are going to concentrate on Amazon GameLift. First, you must create an AWS account. You can get started with a free account. Here are the steps to create your account:

1. Point your browser to `https://aws.amazon.com`.
2. Click the **Create an AWS Account** button. It will be located in the upper-right corner of the screen. Alternatively, you can select the **Create a Free Account** button.

Due to the rapidly changing nature of AWS, the web page you see might differ slightly from what is presented here.

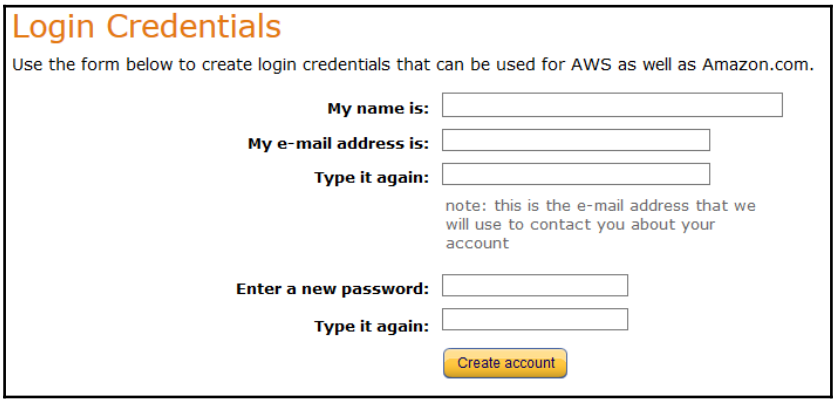


3. Enter your e-mail address or mobile number and select the **I am a new user.** option:



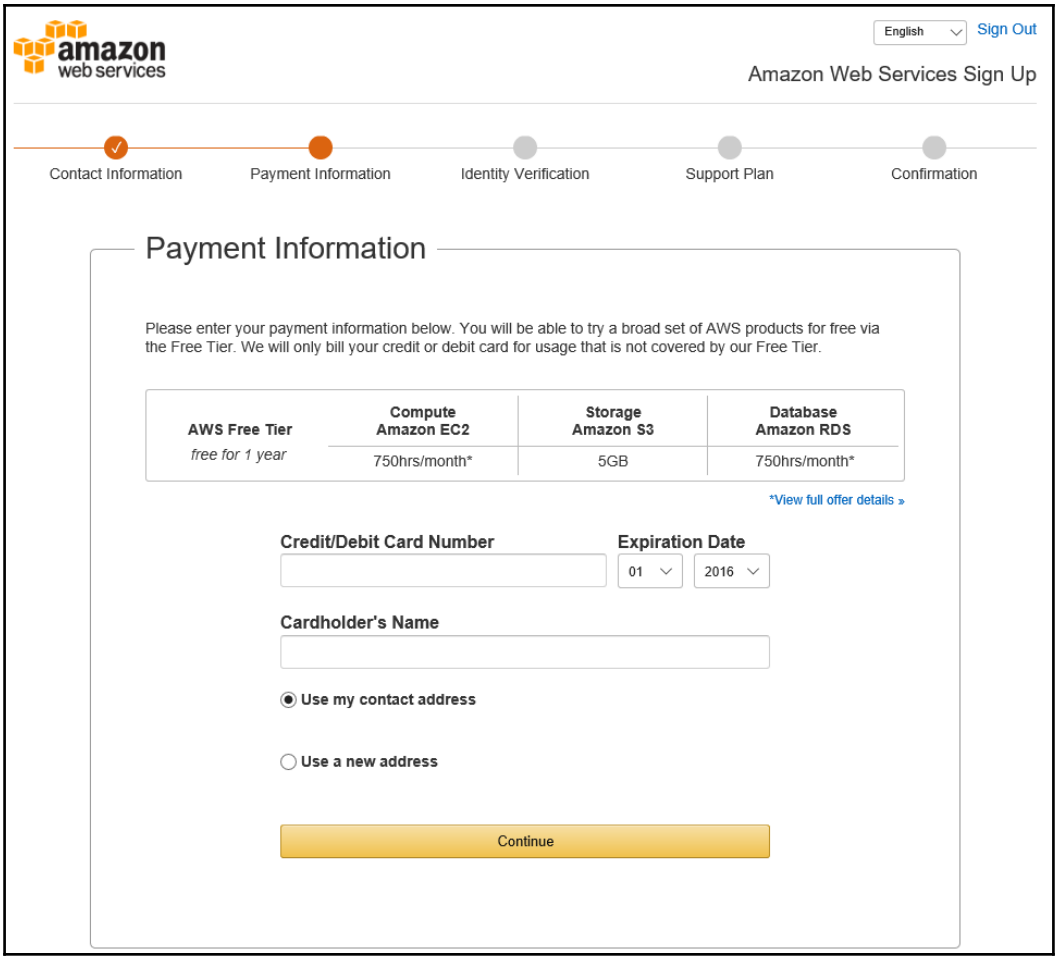
4. Next, you will click the **Sign in using our secure server** button:

5. Next, you will fill in your name, e-mail address, and password for this account. As with any account, you should ensure you have a strong password and protect your login credentials from others:

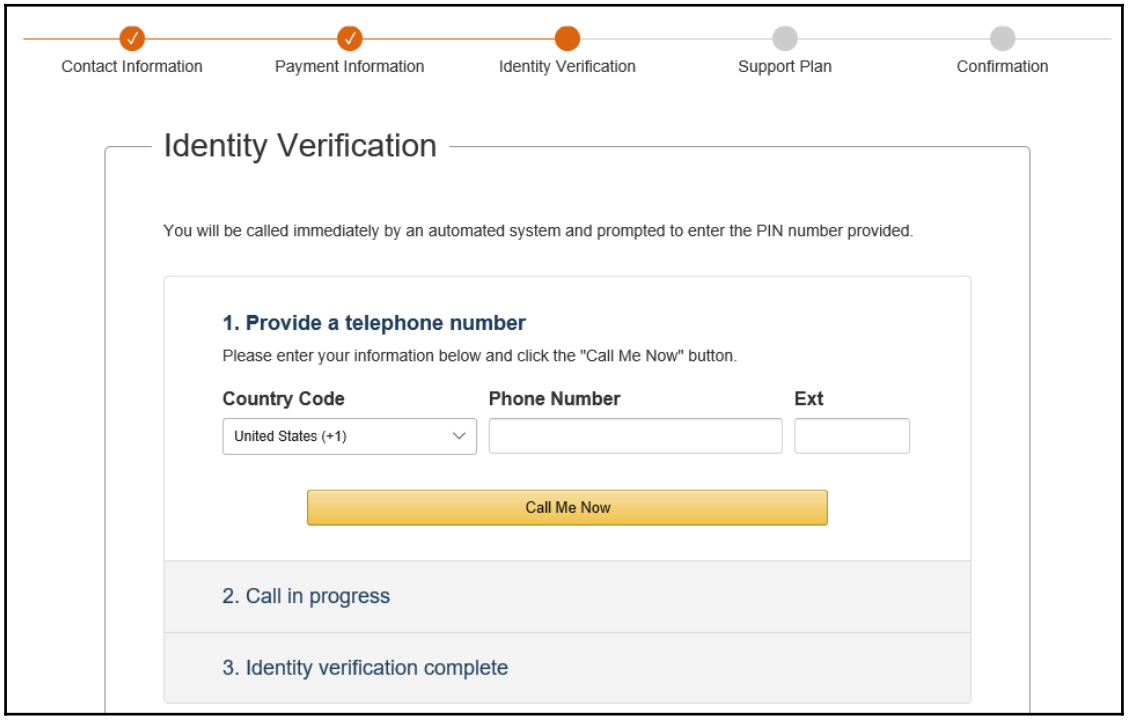


6. The next screen is the **Contact Information** form. You are required to indicate whether your account is a **Company Account** or **Personal Account**. Make that selection and fill in the rest of your contact information.
7. Read the **AWS Customer Agreement** linked on the bottom of the **Contact Information** form. If you agree, click the box to indicate your agreement.
8. Click the **Create Account and Continue** button.

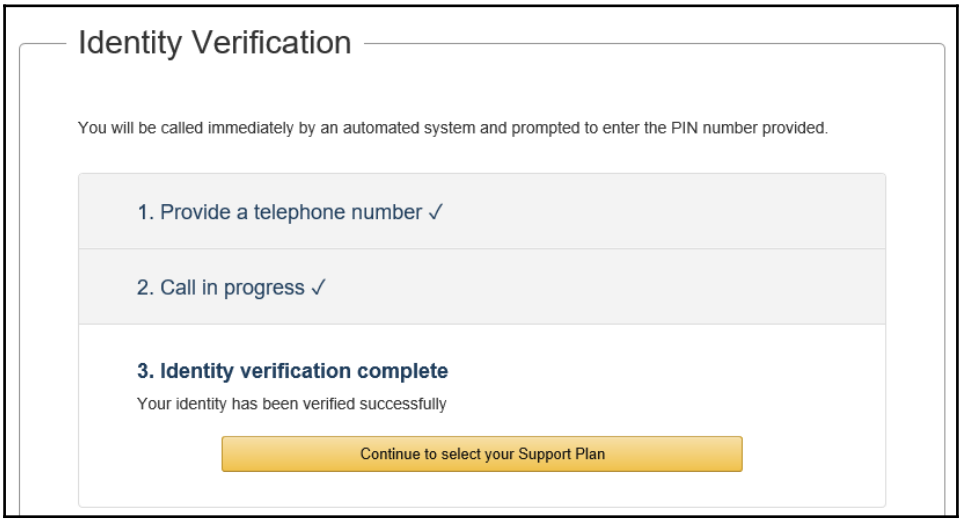
9. The next step is to enter your payment information. You are not paying for anything at this point. As you can see in the following screenshot, the default is **AWS Free Tier**. When you finish entering your payment information, click the **Continue** button. This button might appear as a **Securely Submit** button:



10. Next, you will need to verify your identification. Using the form provided, you will enter your phone number and click the **Call Me Now** button. That will result in a code being displayed on the screen. Amazon will call you using an automated system. You will enter the code into your phone:



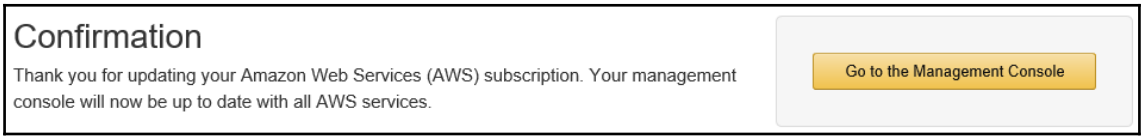
11. Once your call is completed, you will see that both the **Provide a telephone number** and **Call in progress** steps are checked. You will next click on **Continue to select your Support Plan**:



12. At this point, you will need to make a decision on what type of service you want. There are basic, developer, business, and enterprise accounts. The basic account is free and is a great option if you are just getting started. Make your selection and click the **Continue** button. If you selected the basic service, the **AWS Free Tier**, there will be no cost to you for signing up for this service.

It is important to understand that Lumberyard is provided by Amazon for free to developers. While access to AWS is free, at the basic level, use of storage, computing, and other services come at a cost. To see the full list of AWS free and paid services associated with the **Free Tier**, review the information at `aws.amazon.com/free`.

13. Once you complete the signup process, you will receive an on-screen confirmation message, as shown here:



Becoming familiar with the AWS Management Console

1. The **AWS Management Console** provides you with management access to your web services. You can also click on any of the services to learn more and start using them.

A precise URL cannot be provided because AWS is distributed by region. At the time of publication, there are 10 regions.

2. Our interest in the AWS Management Console is with GameLift. Access to that service is listed under the **Game Development** category. If you only see icons under the **AWS Services** section, you can click the **Game Development** icon and then select **GameLift**. You can toggle views by clicking **SHOW ALL SERVICES** or **SHOW CATEGORIES** next to the **AWS Services** title:

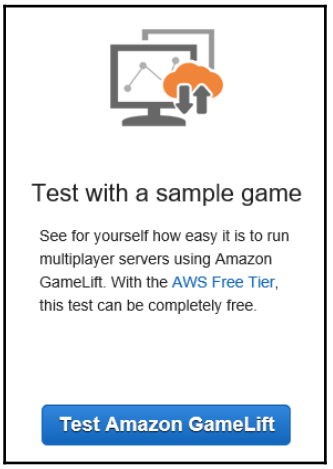
3. When you click on the GameLift link, you are taken to the Amazon GameLift page. We will explore GameLift in the next section.

If you want to learn more about the AWS Management Console, the documentation is located at: `http://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/getting-started.html`.

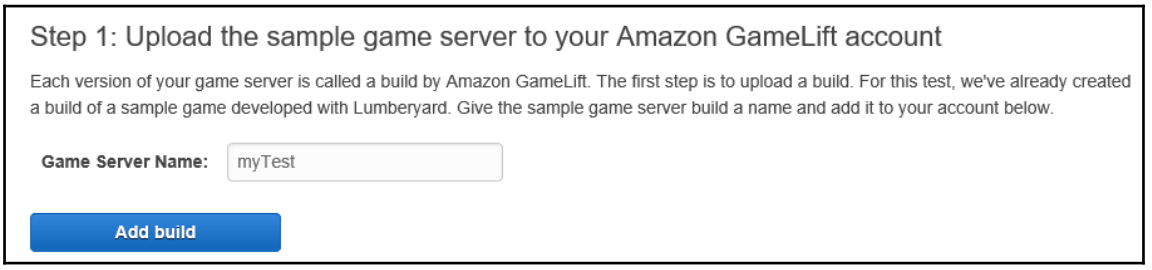
Getting started with Amazon GameLift

Amazon GameLift is a service associated with Lumberyard that allows us to deploy and run scalable multiplayer games. As you'll see in this section, getting started with GameLift is relatively easy:

1. At the Amazon GameLift landing page, click the **Test Amazon GameLift** button. If you are not at the Amazon GameLift landing page, review the steps in the previous section:

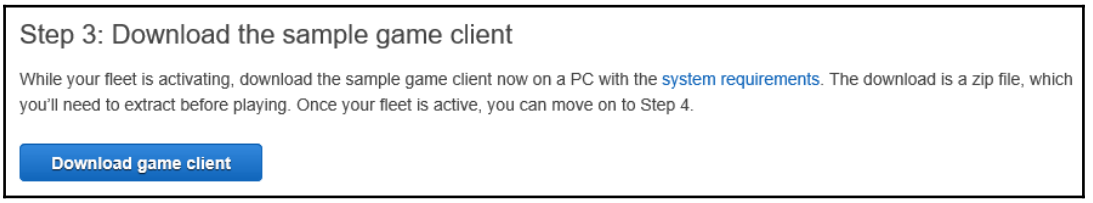


2. On the next page, you will see a five-step schema for testing GameLift using a sample game. Amazon has made this very easy for us and already has a multiplayer game for us to test with. So, your first step is to enter a name in the **Game Server Name** field and click the **Add build** button:

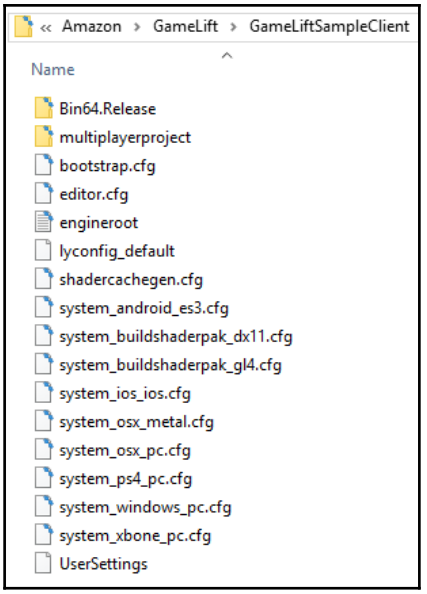


3. When **Step 1** (**Upload the sample game server to your Amazon GameLift account**) is completed, the blue **Add build** button will change to a green **Build added** button. You are now ready for step 2.
4. The next step, **Step 2** (**Deploy the sample game server**), simply requires you to click the **Create a fleet** button. Do that now:

5. This process can take 30-40 minutes so do not be worried if the site seems stuck with **Creating instance…** on screen. During the process you will first see an **Initializing…** message, followed by a **Downloading build…** progress indicator. Once the build is completed, the blue **Create a fleet** button will change to a green **Fleet generated** button.
6. Next, we need to download the game client. We simply click the **Download game client** button:



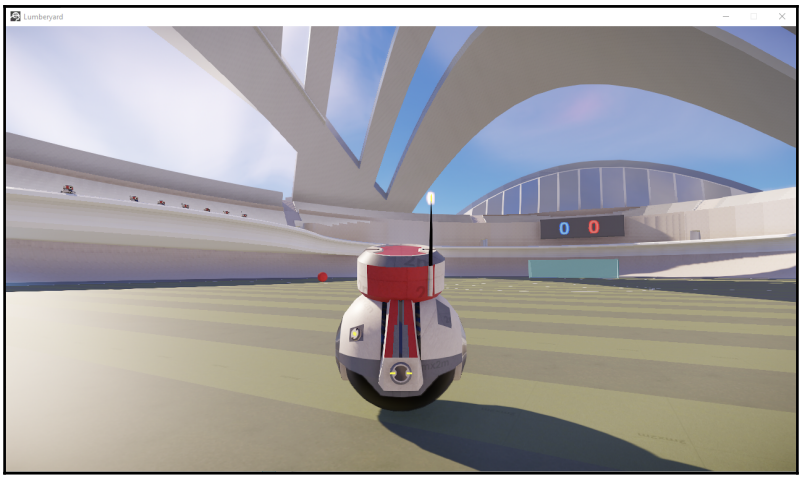
7. Once the zip file is downloaded, extract all files. You will want to remember where you placed the files. The following screenshot shows the extracted files placed in the `Amazon\GameLift\GameLiftSampleClient` folder:



8. Click the **Generate Token** button. GameLift will generate a `Player Session Token`. Copy that token. You will use it in the next step.

The `Player Session Tokens` expire after one minute. This is a necessary limit to preserve resources. If your token expired, you will see an error message when trying to connect. In that case, you will need to return to the GameLift console and generate a new token.

9. Our next step is to connect the downloaded game client to the sample game server, then test the game. Open the `Bin64.Release` folder and launch the `MultiplayerProjectLauncher.exe` file.
10. Paste the `Player Session Token` in the text box and click the **Connect** button. You will now be able to play the game:



11. At this point, you can generate additional `Player Session Tokens`, up to a total of eight. This allows you to test the multiplayer game with multiple players.

Checking performance with the GameLift Dashboard

Now that you have tested a multiplayer game with GameLift, you can see how your game server performed. The Amazon GameLift Dashboard provides insights to that performance.

In the previous steps, you generated a test game server build, deployed the sample game server, downloaded the sample game client, generated `Player Session Tokens`, and play-tested the game with one or more players. At the end of those steps, GameLift presents you with a **View dashboard** button. This gives us access to the **GameLift Dashboard**.

The **GameLift Dashboard** displays all the **Fleets** you generated and provides details about each. Here is a screenshot of the **Fleet** we created in the previous section. As you can see, there is a single active fleet with one active instance:

Status	Fleet ID	EC2 type	Active instances	Protection	Active game sessions	Current player sessions	Uptime	Date created
Active	fleet-26f2d56a-0b8a-40dd-871e-b8682a914d1a	c3.large	1	No protection	1	0 of 8	00d 15h 37m 21s	Sun, 26 Jun 2016 0:43:0

The next section of the **GameLift Dashboard** has eight tabs. Information provided on each tab is detailed in the following table:

<table>
<tr><th>Tab</th><th>Contents</th></tr>
<tr><td>Metrics</td><td>Provides a graph showing game and hardware data on a timeline. This information is filterable.
<table>
<tr><td>Game</td><td>Available player sessions
Current player sessions
Player session activations
Active game sessions
Activating game sessions</td></tr>
<tr><td>Hardware</td><td>CPU utilization
Network in
Network out
Disk read bytes
Disk write bytes
Disk read operations
Disk write operations</td></tr>
</table></td></tr>
<tr><td>Events</td><td>Here you will see a list of all events relevant to your fleet.</td></tr>
<tr><td>Scaling</td><td>The scaling tab gives you an insight into scaling information. The default is to allow GameLift to perform auto-scaling, but you can also create your own scaling rules on this tab.</td></tr>
<tr><td>Game sessions</td><td>When a game is active, you can view detailed information about each session.</td></tr>
<tr><td>Build</td><td>Metadata for your build.</td></tr>
<tr><td>Launch (also known as Capacity Allocation)</td><td>Shows the file system path for the launch file, as well as launch parameters.</td></tr>
<tr><td>Ports</td><td>You can view and modify network port settings.</td></tr>
<tr><td>Logs</td><td>Indicates the file system path for your logs.</td></tr>
</table>

Summary

In this chapter, we examined multiplayer games in the context of creating them with Amazon Lumberyard. We started with a review of multiplayer game design and development considerations and challenges. You were introduced to AWS and Amazon GameLift. You gained hands-on experience creating a multiplayer game session using GameLift.

In the next chapter, we will dive back into Lumberyard and review how audio and sound effects can be used to bring your games to life. You will learn about audio architecture and gain hands-on experience adding background audio and sound effects to your games. You will also learn how to import audio assets and how to use the **Audio Controls Editor**.

8

Bringing Your Game to Life with Audio and Sound Effects

In the previous chapter, we thoroughly explored multiplayer game concepts. Our exploration included a review of multiplayer game design, and development considerations and challenges. We examined AWS with a specific focus on Amazon GameLift, the AWS that allows you to deploy, operate, and scale multiplayer games. Rather than theorize, we used a hands-on approach to become familiar with Amazon GameLift.

In this chapter, we will explore Lumberyard's **Audio System**. We will take a look at the complexity of Lumberyard audio and examine the components of the Lumberyard Audio System. We will revisit the Amazon Lumberyard download web page and obtain examples for our exploration of Lumberyard's audio features.

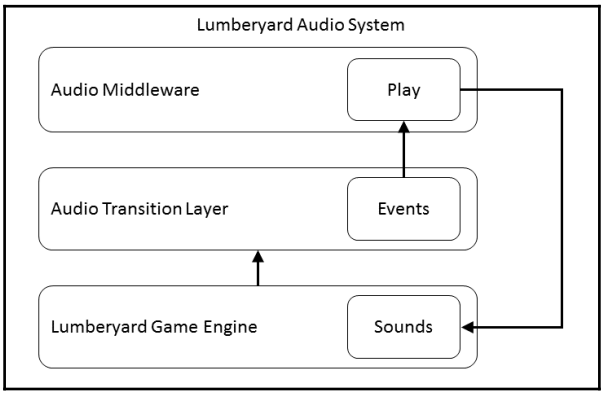
After reading this chapter, you will:

- Understand the Lumberyard Audio System architecture
- Install and use the Wave Works Interactive Sound Engine
- Download and import the BeachCity Asset Package
- Download the **Legacy Game** sample
- Understand how to create **Audio Areas**
- Understand how to create **Audio Triggers**
- Understand the **Audio Controls Editor**

Getting started with the Lumberyard Audio System

Lumberyard's method of handling in-game audio is a bit complex. Audio assets are not created in Lumberyard; they are created using the *Wwise* software. Wwise is an acronym for Wave Works Interactive Sound Engine by Audiokinetic. There are two versions of Wwise: a full, for-pay, version and a free, lite, Wwise LTX version.

Game audio works through collaboration and communication between Lumberyard, the Audio Transition Layer, and Wwise. When an in-game event requiring audio is triggered, Lumberyard notifies, using the Audio Transition Layer, the audio middleware to play the sound. The following diagram graphically illustrates a high-level view of the audio system pipeline:



Lumberyard's audio pipeline might seem unnecessarily complex, especially when compared to how other game engines implement their audio systems. This complexity results in great efficiencies and mitigates frustrations due to frequent changes. To explain, since the Audio Transition Layer is used, audio asset changes do not require any changes or updates to the game. This represents a tremendous benefit for game developers.

Audio asset basics

There are several concepts and terms that are important to understand when dealing with game audio using Lumberyard. The following terms relevant to audio in Lumberyard are presented in alphabetical order for ease of reference:

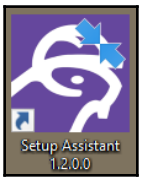
- **Ambiance**: Background noise.
- **Audio library**: A grouping of audio assets with XML files to indicate associations with Lumberyard, the Audio Transfer Layer, and the Audio Middleware.
- **Audio pipeline**: A linear sequence of steps and events that result in audio being played in a game made with Lumberyard.
- **Sound bank**: A set of audio files, which include metadata.
- **Trigger**: An in-game event or occurrence that causes a reaction. Examples include stepping on or passing over an object, colliding with an object, leaving an area, making decision tree decisions, placing objects, timer-related events, and so much more.

Wave Works Interactive Sound Engine

As previously discussed, there are two versions of Wwise: a full, for-pay version and a free, lite version. We will implement the lite version (Wwise LTX).

Wwise LTX is only available through Lumberyard and is part of the initial game engine download. The following steps will guide you through the installation of Wwise LTX.

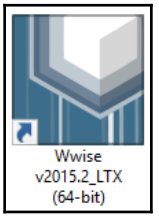
1. Launch the **Lumberyard Setup Assistant**. The **SetupAssistant.exe** file should be located in the `\Bin64` subfolder. You can also identify the program from its icon, shown here:



2. In the **Lumberyard Setup Assistant** dialog window, click the **Install software** link in the left navigation pane.
3. In the main view pane, under **Optional software**, click the **Get it** link to the right of the **Audiokinetic Wwise LTX Authoring Tool** row.

4. The **Wwise LTX Setup** dialog will appear. If you already have the software installed, click the Cancel button; otherwise, complete the installation process:

5. Close the **Lumberyard Setup Assistant**.
6. Now that you have Wwise LTX installed on your computer, let's launch the app and create a project. We are creating a project to demonstrate the required steps. You can use these steps each time you need to create an audio project:
7. Navigate to the Wwise LTX app and launch it:

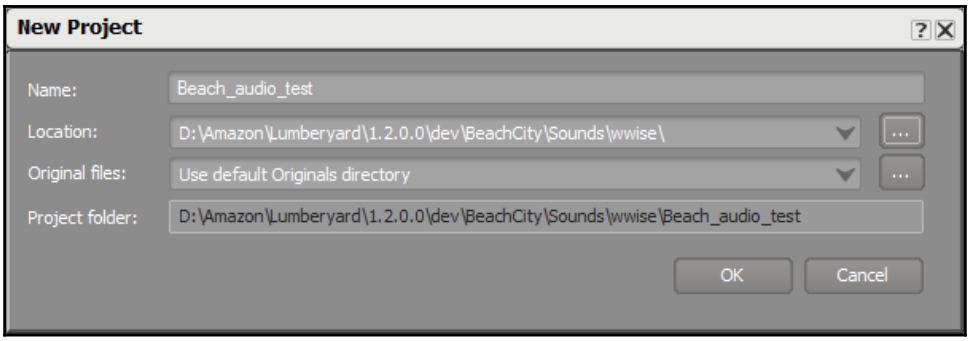


8. The **Wwise LTX Project Launcher** will appear. Click the **New** button.

If you have not yet accepted the **End-User License Agreement**, you will be prompted to do so before proceeding any further.

9. Enter a name for your project, indicate where you want the project files saved, then click the **OK** button. The following image is representative of where in a project's file path the project should be created.

Each Lumberyard game will have a single Wwise project associated with it. That project must be located where Lumberyard expects to find it.



10. The Wwise LTX application can be overwhelming for those not experienced with sound engines. The application has in-app documentation that can be referred to for help using the engine.

Using sample asset packages

When learning to use a new game engine, such as Lumberyard, it can be beneficial to use sample projects and asset packages for testing and familiarization. For audio testing, you can use almost any package, or even create your own level from scratch. Downloading and installing asset packages is more complicated than it seems.

The following section walks you through the process of downloading, installing, and configuring an asset package for use. This is an optional section that will result in the `BeachCity Asset` package being installed on your computer. The installation process is lengthy and the hard drive space requirements are large.

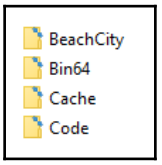
BeachCity Asset package

In order to see how audio is implemented in a game, let's review an existing game example. The following steps will guide you through the process of downloading and installing the `BeachCity Asset` package and configuring Lumberyard to use it:

1. In a browser, navigate to the **AWS Lumberyard** download page: `https://aws.amazon.com/lumberyard/downloads`.
2. Scroll down to the **Beach City Asset Package** and click the **Download** button.

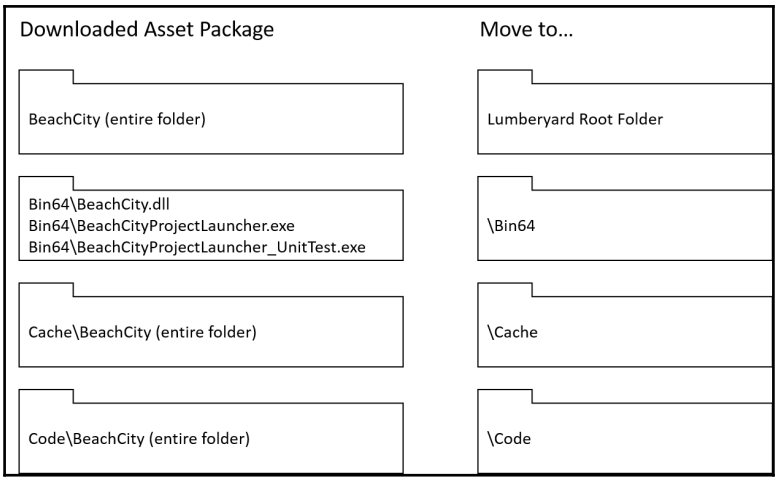
The download process can take a while, depending on your Internet bandwidth. The compressed file is several gigabytes. When uncompressed, the files are approximately 15 gigabytes.

3. Once your download has completed, unzip the asset package to a temporary folder. When the file extraction process is completed, you will have four folders inside the `\dev` folder, as illustrated here:

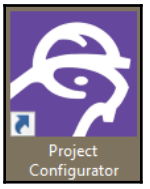


Just like with the download, the file extraction process can take several minutes.

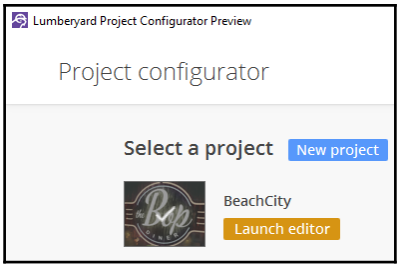
4. Move the `BeachCity` folder into your Lumberyard root folder. The root folder should be similar to `\Amazon\Lumberyard\1.2.0.0\dev`.
5. Next, move the contents of the `Bin64` folder from your temporary folder into Lumberyard's `Bin64` folder. The `Bin64` folder is located in the root folder. You will be moving three files: `BeachCity.dll`, `BeachCityProjectLauncher.exe`, and `BeachCityProjectLauncher_UnitTest.exe`.
6. Move the `Cache\BeachCity` folder into Lumberyard's `Cache` folder, located in the root level.
7. Next, move the `Code\BeachCity` folder into Lumberyard's `Code` folder, located in the root level. At this point, all of your files should have been moved from the extracted files to Lumberyard's file structure. The following figure illustrates the files you should have moved:



8. Launch the Lumberyard **Project Configurator**:



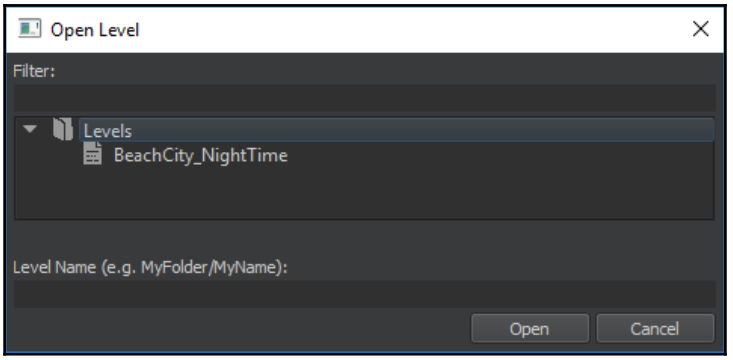
9. You should see **BeachCity** listed in the **Project Configurator**. Click the **BeachCity** icon, click the **Set as default** button, and then, depending on your Lumberyard version, click the **Launch editor** or **Enable Gems** button. This will result in Lumberyard's **Asset Processor** instantiating and processing the assets in the **BeachCity Asset Collection**. This process is referred to as a project rebuild.



10. Once the Asset Processor has completed, close the **Project Configurator** interface.

The `BeachCity Asset` package is very large; it could take the Asset Processor well over an hour or more to process the entire package. Do not interrupt the process.

11. Launch the **Lumberyard Editor**.
12. Click the **Open level** button in the **Welcome to Lumberyard Editor** dialog window. If you have this window suppressed, use the **File** | **Open** menu option.
13. You are now presented with the **Open level** dialog window. Navigate to and select the **BeachCity_NightTime** level. Then, click the **Open** button.



The **BeachCity_NightTime** level is now available in Lumberyard. It is not playable, but can be configured for actual gameplay. That is beyond the scope of this chapter and is a good source of post-chapter self-study.

Legacy Game Sample

In this section, we will download the Legacy Game Sample available at the Lumberyard download page: `https://aws.amazon.com/lumberyard/downloads/`.

1. Click the **Download** button in the **Legacy GameSample** section of the Lumberyard download page.
2. Extract all files from the downloaded ZIP file.
3. Open two File Explorer windows, one with the Lumberyard root folder, the other with the extracted files.

4. Move all the files in the downloaded `\Bin64`folder to the `\Bin64` folder in the Lumberyard root folder.
5. Move all the files in the downloaded `\Code` folder to the `\Code` folder in the Lumberyard root folder.
6. Move the `GameSDK` folder from the download to the `\dev` folder that is the Lumberyard's root folder.
7. The Legacy Game Sample uses the **Woodland Asset Package**, which must also be downloaded. In a browser, navigate to the AWS Lumberyard download page, `https://aws.amazon.com/lumberyard/downloads`.
8. Scroll down to the **Woodland Asset Package** and click the **Download** button.

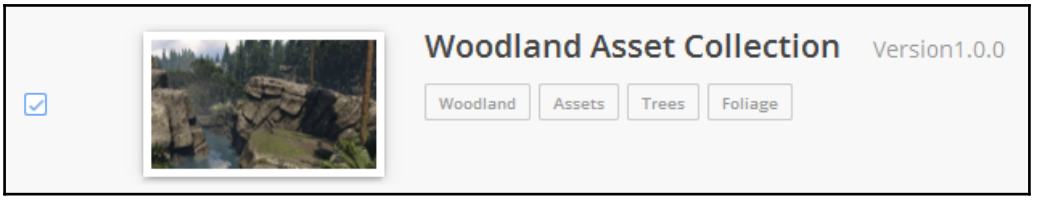
The download process can take a while, depending on your Internet bandwidth. The compressed file is several gigabytes.

9. Once your download has completed, unzip the compressed file. Navigate to `\dev\Gems` and move the `AssetCollection_Woodland` folder from the newly unzipped hierarchy to the `\Amazon\Lumberyard\1.2.0.0\dev\Gems` folder.
10. Launch the **Lumberyard Project Configurator**.
11. Click on the **GameSDK** project:

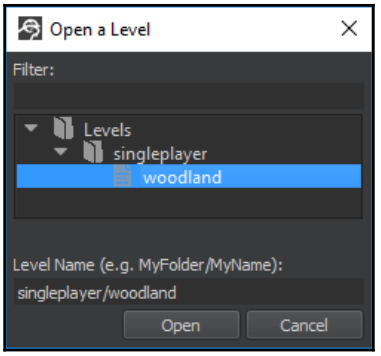
12. Click the **Set as default** button located in the upper-right corner of the dialog window. This will result in Lumberyard's Asset Processor instantiating and processing the assets for this project.
13. Click the **Enable packages** link in the upper-right corner of the **Project Configurator** interface:

If you are using Lumberyard 1.3 or later, the **Enable packages** link is replaced by an **Enabled Gems** link under the **Project**:

14. Scroll down until you find the **Woodland Asset Collection**. Check the box to the left of the asset icon:



15. Click the **Save** button in the upper-right corner of the interface. This will result in Lumberyard's Asset Processor instantiating and processing the assets in the **Woodland Asset Collection**. This process is referred to as a project rebuild.
16. Once the Asset Processor has completed, click the **Back to projects** link located in the top-left corner of the **Project Configurator** interface.
17. In the **Lumberyard Configurator**, select the **Launch editor** button underneath the **Game SDK** icon. Alternatively, if that button is not present, close the **Lumberyard Configurator** and launch the **Lumberyard Editor**.
18. Open the **woodland** level:



If you receive errors when loading the level, it is likely okay. The engine is still in beta and the legacy package is not receiving updates. You should be able to proceed, despite any errors you receive.

19. This is a full game level, so you are able to play the game once it loads. In the next section, we will review the game audio.

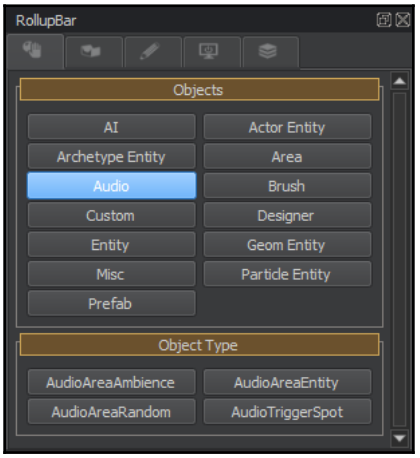
Audio options

There are several ways we can add audio to our Lumberyard games: Audio triggers, ambient or background audio, and sound effects. We will take a look at these three methods next.

Audio triggers

Our next task is to add an **Audio Trigger Spot** to the level. Here are the steps:

1. In the **RollupBar**, select **Objects** | **Audio** | **AudioTriggerSpot**:



2. Next, click the **Entity: AudioTriggerSpot** bar from the **RollupBar** to your level. Clicking in the Viewport adds an **Audio Trigger** to the level. You can review the settings in the **RollupBar** and edit the script using the **Edit Script** button. If you do not see the **Edit Script** button, click the **Entity:AudioTriggerSpot** bar.

Ambient (background) audio

We can add ambient or background sounds to our Lumberyard games. To do this, we will identify an area in the game where the audio can be heard. There are fade levels associated with ambient sounds, so the closer the player gets to the identified area, the louder the sound level will be:

1. Our first step is to define a shape using the **RollupBar**, select **Objects** | **Area** , and then select **Object Type** | **AreaBox**, **AreaSphere**, or **Shape**.
2. Click in the Viewport to place the area object.
3. With the selection made, the parameters can be set to include the size, shape, and location of the area. You simply click in the Viewport to place the `Area` object.
4. Next, select **Objects** | **Audio** | **AudioAreaAmbience**.
5. Click in the level to place the **Entity: AudioAreaAmbience**.
6. In the **AudioAreaAmbience Properties** area, click **PlayerTrigger**, click the folder icon to the right of **PlayTrigger**, and navigate to an audio file. You will see several audio files in the **l_woodland** | **l_woodland_amb** file folder. Once you've selected an audio file, click the **OK** button.
7. Click **Rtpc** and then click the file folder icon to the right of the **Rtpc** row. Navigate to and select the **g_atl_rtpcs** | **environment** | **area_fade_distance** item and click the **OK** button.
8. Next, we need to link the ambient sound to the area we previously created. Click the **Entity Links** area of the **RollupBar** and click the **Pick Target** button.
9. Now, select the area in the Viewport. You should see the entity linked in the **Entity Links** pane.
10. Switch to Game Mode to test the background audio.

Adding sound effects

One of the ways to improve the immersiveness of our games is to include sound effects. For example, when a gun shoots a bullet, we should hear a sound. Let's add this sound to the test game now:

1. Using the top menu system, select **View** | **Open View Pane** | **Geppetto**. This opens the **Geppetto Editor**.
2. In the **Assets**pane, select **Animations** | **Weapons** | **Pistol** | **1p** | **stand_tac_recoil_pistol_iron_add_1p_01**.
3. In the **Properties** pane, select the down arrow to the right of **Animation Events** and select **insert**.
4. On the row that was just created, click the down arrow icon and a select sound.
5. Click the empty field to the right of the down arrow and select **w_pistol** | **w_pistol_fire_fp** | **Play_w_pistol_fire_fp**.
6. Click the **OK** button.

Using source code for sound effects

Another method of implementing sound effects in our games is to edit the appropriate source code. An example code snippet is:

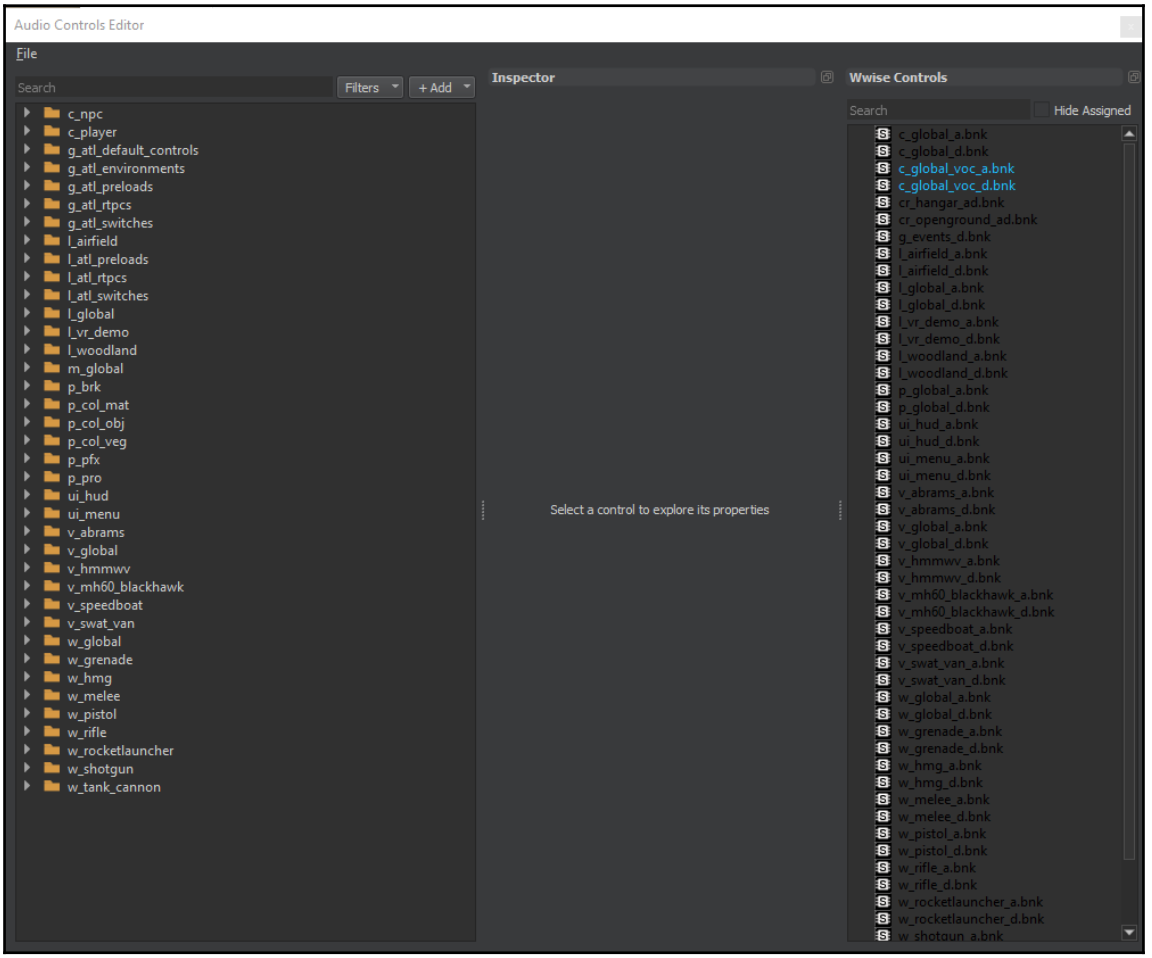
```
<Effect name ="pistol_shot">
<Audio Trigger'="Play_pistol_shot" />
</Effect>
```

This code tells the game engine to play the audio trigger **Play_pistol_shot** when the Effect is triggered.

You can accomplish the same function by creating a Flow Graph. Creating a Flow Graph will result in the source code being built for you.

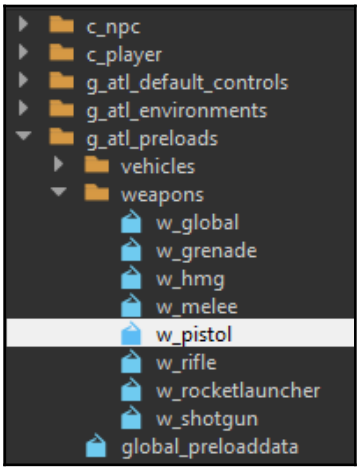
Audio Controls Editor

The **Audio Controls Editor** is accessible using the **View** | **Open View Pane** | **Audio Controls Editor**. This editfor is where you will map your game entities, such as actions and events, to the audio system:

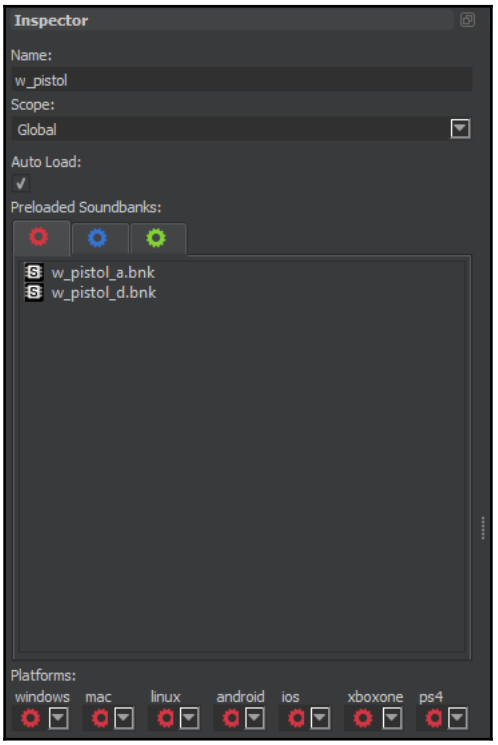


The **Audio Controls Editor** has a three-column layout. The first column is presented in hierarchical form and is the **Audio Translation Layer** panel. The second and third columns are the **Inspector** and **Middleware Controls** panels, respectively.

You can navigate to specific items in the **Audio Translation Layer** panel by clicking the gray triangles to the left of the folder icons. When the triangle points to the right, it can be clicked to expand the section. This results in the triangle pointing downwards:



When an item is selected in the left-most column, the item's metadata is displayed in the **Inspector** pane:



The **Inspector** pane displays the item's name and scope. There is also an **Auto Load** checkbox that can be selected or de-selected. The individual sounds are listed in the **Preloaded Soundbanks** area of the **Inspector**. There are three colored cogs (red, blue, and green), each with its own tab. As you can see in the **Platforms** section, you can have three different sound banks and specify which one is associated with each distribution platform (Windows, Mac, Linux, Android, iOS, Xbox One, and PS4).

The third column, the **Middleware Controls** pane, lists the sound banks that are part of the Audiokinetic Wwise LTX project associated with your game.

Sound banks are groups of audio files.

Summary

In this chapter, we scratched the surface of audio concepts regarding Amazon Lumberyard games. We looked at Lumberyard's Audio System architecture. We also reviewed the Wave Works Interactive Sound Engine. Following that review, we installed the Wwise LTX software. To make our exploration of the audio system more in-depth, we downloaded and installed asset packages and the **Legacy Game Sample** from the Lumberyard website. We concluded our review of Lumberyard audio with a look at audio areas, audio triggers, and the **Audio Controls Editor**.

In the next chapter, we will look further into AWS as they apply to Lumberyard. Specifically, we will look at **Cloud Canvas**, **Cloud Storage**, **Simple Queue Service**, and **Simple Notification Service**.

9
Employing Cloud Computing and Storage

In the previous chapter, we examined how Lumberyard handles audio events in games. In addition to simply playing sounds, we reviewed Lumberyard's Audio System architecture, which includes audio areas, audio triggers, and the Audio Controls Editor. We also reviewed Wwise by Audiokinetic. We used example asset packages and a sample game to make our audio review contextual.

In this chapter, we will turn back to AWS. You'll recall that, in `Chapter 7`, *Creating Multiplayer Gameplay*, we used GameLift, one of the many AWS. For this chapter, we will review two additional Web Services: **Cloud Canvas** and **Amazon Simple Storage Service**.

After reading this chapter, you will:

- Understand the need for cloud-based solutions for your Lumberyard games
- Use the **AWS Console** to create an **Identity and Access Management** (**IAM**) user
- Understand how to use **Cloud Canvas**
- Understand how to enable the **Cloud Canvas** gem
- Understand how **Amazon S3** is accessed

The need for cloud-based solutions

In `Chapter 7`, *Creating Multiplayer Gameplay*, we created a free AWS account. We will use that account for this chapter.

If you skipped `Chapter 7`, Creating Multiplayer Gameplay, please follow the steps in the *Amazon Web Services* section before proceeding with this chapter.

AWS is a family of cloud-based solutions offered by Amazon, collectively known as AWS. Lumberyard and AWS are inexorably linked, which results in great benefits for the AAA game developer (that's you!). When we develop multiplayer games, we need a cloud-based solution for networking, storage, notifications, leaderboards, and so much more. AWS has a suite of offerings that we can use in our games.

Remember, cloud computing solutions, including those associated with AWS, are not free. Pricing is usually based on usage and not static plans. This allows you to pay for what you use, and not for what you do not use. It is recommended that you continually review your account so that you do not incur any costs you did not anticipate.

When reviewing the various AWS offerings, it can seem overwhelming. That is due to Amazon's increasingly impressive family of cloud-based services. Our use of AWS is to help support the games we build with Lumberyard. At the core, AWS can provide us with backend servers, hosting, storage, and processing – all as needed and scaled to changing requirements. These combined offerings can be considered your game's infrastructure. Keep this in mind as we review specific offerings.

You have already been exposed to GameLift in `Chapter 7`, *Creating Multiplayer Gameplay*. Here, we will cover two additional AWS offerings: Cloud Canvas and **S3**. Each offering will be discussed in depth later in this chapter.

Cloud Canvas overview

As the name suggests, Cloud Canvas is a visual interface to AWS. In fact, it is a visual scripting interface, much like Flow Graph is with Lumberyard. As you'll see later, this interface is used to build connections between your game and web services.

Some of the stated benefits of using Cloud Canvas include:

- Incorporate cloud-based features in your games
- Use Flow Graph to connect to AWS services
- Implement backend/server-based features such as leaderboards
- Use visual scripting to create functionality

Amazon S3 overview

Amazon S3 is short for Amazon Simple Storage Service. This core AWS offering provides our Lumberyard games with scalable cloud storage. The name suggests it is a simple service; in fact, it is simple to use. The complexities are taken care of by the service and included storage classes and security.

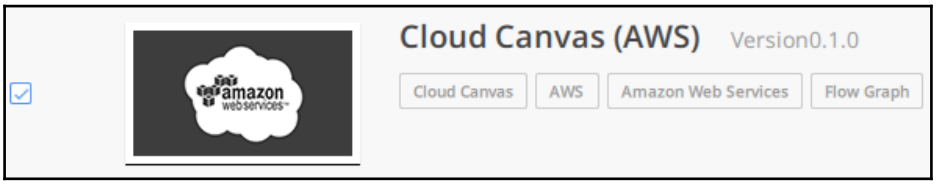
An AWS account is required to use the features covered in this chapter. See `Chapter 7`, Creating Multiplayer Gameplay for details.

Cloud Canvas in action

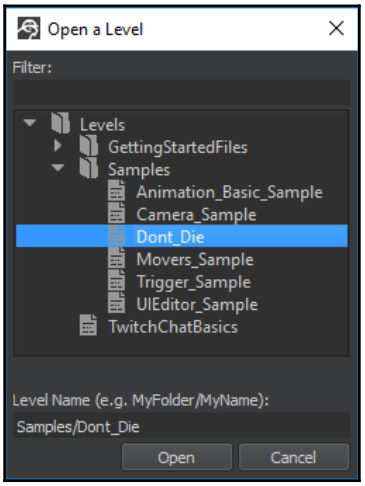
Let's use a game example to help us understand how to enable and use Cloud Canvas for our games. Here are the steps:

1. Launch the **Project Configurator**.
2. Click the **SamplesProject** icon:

3. Next, you'll want to ensure Cloud Canvas is enabled, so click the **Enable Gems** link.
4. Scroll down the page until you find the **Cloud Canvas (AWS)** row. Ensure the checkbox is checked:



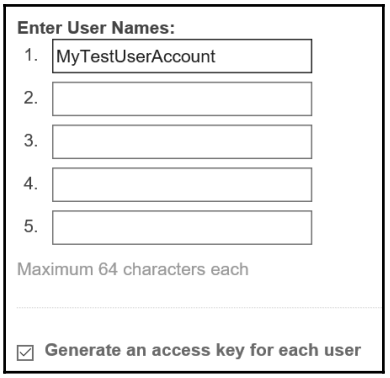
5. Click the **Back to Projects** link in the top-left corner.
6. Click the **Set as Default** button in the top-right corner. This will result in the Asset Processor converting all the appropriate objects for use in your game.
7. Once the Asset Processor has completed its operation, launch the **Lumberyard Editor**.
8. Click the **Open level** button, and navigate to and select the **Dont_Die** level. Then, click the **Open** button:



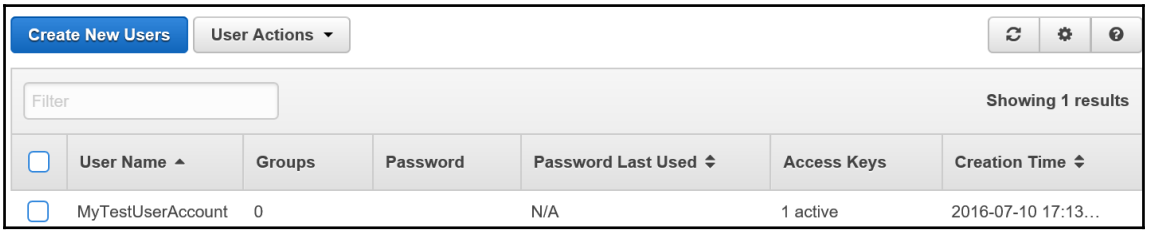
If you were to try and play the game, you would receive a missing authentication token error. That's okay; we'll take care of that next.

9. Log in to your AWS Console to create an **Identity and Access Management** (**IAM**) user. You can use this direct link: `https://console.aws.amazon.com/iam/`.
10. At the **IAM Console**, select the **Users** link either in the left navigation pane or the center column. Both links take you to the same location.
11. Click the **Create New Users** button.

12. Enter a user name in the first input box and click the **Create** button:



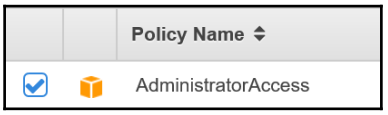
13. Click the **Download Credentials** button. This will download your security credentials in a CSV file.
14. After you have verified that your credentials downloaded, click the **Close** link located in the bottom-right area of the page (to the left of the **Download Credentials** button). You should see your account listed, as shown in following screenshot:



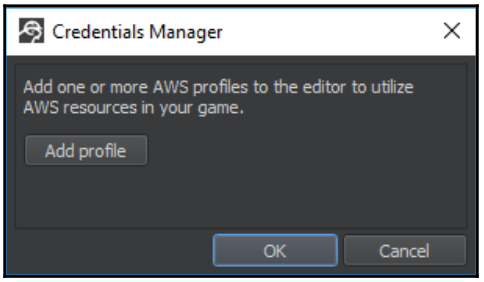
15. Click on the user name that you just created.
16. You will now see four tabs (**Groups, Permissions**, **Security Credentials**, and **Access Advisor**). Click the **Permissions** tab:

17. Click the **Attach Policy** button. This will provide you with a list of available policies.

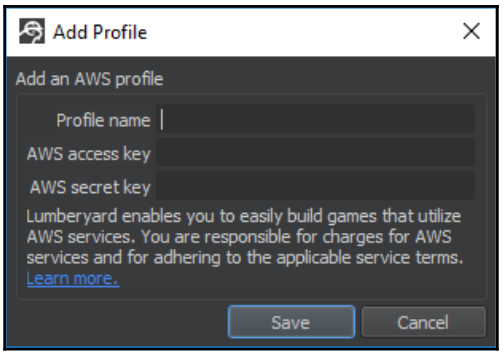
18. Locate the **AdministratorAccess** policy, check the checkbox to the left of that item, and click the **Attach Policy** button:



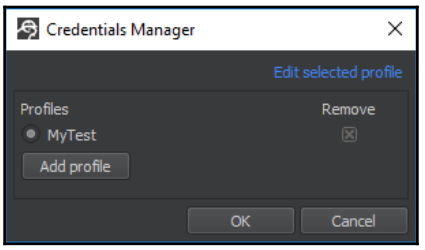
19. Next, click the **Security Credentials** tab. Here you will set up your IAM account password.
20. Click the **Manage Password** button. Follow the instructions to create a password. Click the **Apply** button to finalize the password setup.
21. Click the **Dashboard** located at the top of the left navigation pane. You will now see your **IAM users sign-in link**. Copy that link.
22. Paste the IAM users sign-in link into a new browser window or tab and hit Enter.
23. You are now at the AWS sign-in page. Enter the newly created user name and password. Click the **Sign In** button.
24. Go back to your instance of Lumberyard and click **AWS** | **Credentials Manager** in the top menu bar. This will launch the **Credentials Manager** dialog window.



25. Click the **Add Profile** button. Here you will create a profile name and enter the **AWS access key** and **AWS secret key** from the `credentials.csv` file you created in step *13*. After you've entered your information, click the **Save** button:

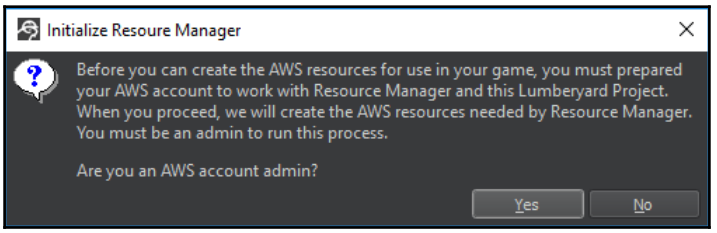


26. The **Credentials Manager** will show the profile you created. Click the **OK** button to close that dialog window:



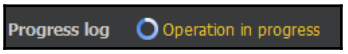
27. Open **AWS** | **Cloud Canvas** | **Cloud Canvas Resource Manager**.

28. In the **Canvas Resource Manager**, click **Resource Groups** | **DontDieAWS** in the left panel.
29. Click the **Create Resources** button. This will result in the **Initialize Resource Manager** dialog window to be displayed. Click the **Yes** button:

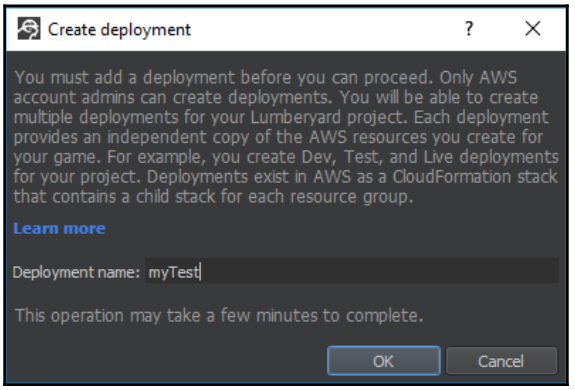


30. You will now see the **Initialize Cloud Canvas Resource Manager** dialog window. Click the **Create** button.

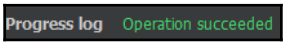
31. In the **Cloud Canvas Resource Manager**, view the **Progress log** section and wait until all resources have been created. You will see an **Operation in progress** message.



32. You will now be prompted to create a deployment. As indicated in the **Create deployment** dialog, a deployment generates an independent copy of the AWS resources for your game. Enter a **Deployment name** in the text field and click the **OK** button:



33. Monitor the system messages in the **Cloud Canvas Resource Manager**. When all processing is complete, you will receive an **Operation succeeded** message onscreen. At this point, you can close the dialog window.



34. Now we will log back in to our AWS account using the user name you created in Step 23. Once you have logged in, you can proceed to the next step.
35. Click **Services** from the top menu banner.
36. Select **CloudFormation**, under **Management Tools**.

AWS **CloudFormation** is a collection of AWS resources. Using **CloudFormation** for a group of related resources improves developer efficiency when updating them.

37. Scroll to the bottom of the page and click the **Create New Stack** button.
38. Select the **Select a sample template** radio button and select **CloudFormer** from the drop-down menu under the **Tools** label.
39. Click the **Next** button.
40. At this point, you are presented with the request for a **Stack name**, **Password**, and **Username**. Going beyond this point could incur charges to your account.

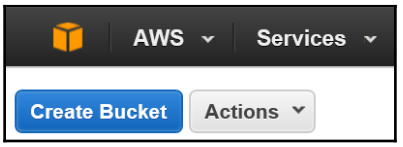
Amazon S3 in action

Earlier in this chapter, we learned that Amazon S3 is the AWS service that provides scalable cloud storage. One of the most useful aspects of this service is that it is scalable. This means we only pay for what is used by our game. We know that the number of players and the amount of time they play our game will be highly variable. That makes scalable cloud storage solutions ideal.

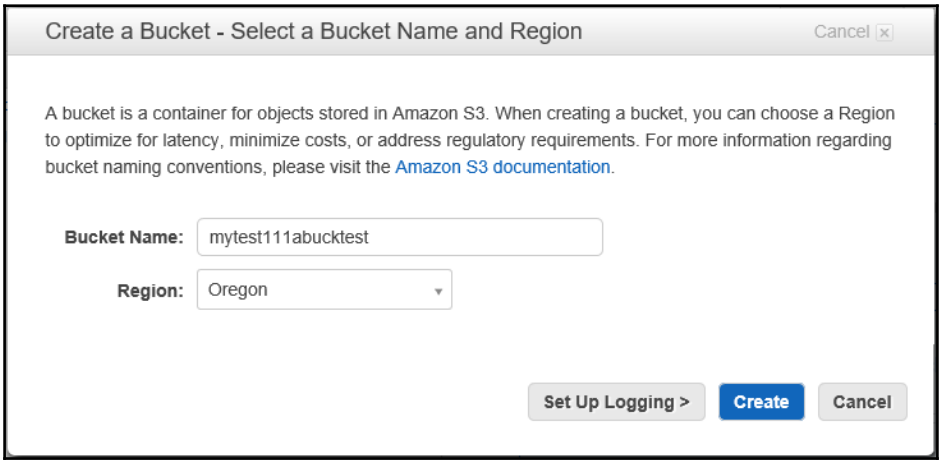
Let's look at how to use the Amazon S3 service:

1. Point your browser to `http://aws.amazon.com/s3/`.
2. Locate and click the **Sign in to the Console** button. This is likely to be located in the rightmost column. You should be taken to the AWS Management Console because you previously created an AWS account.

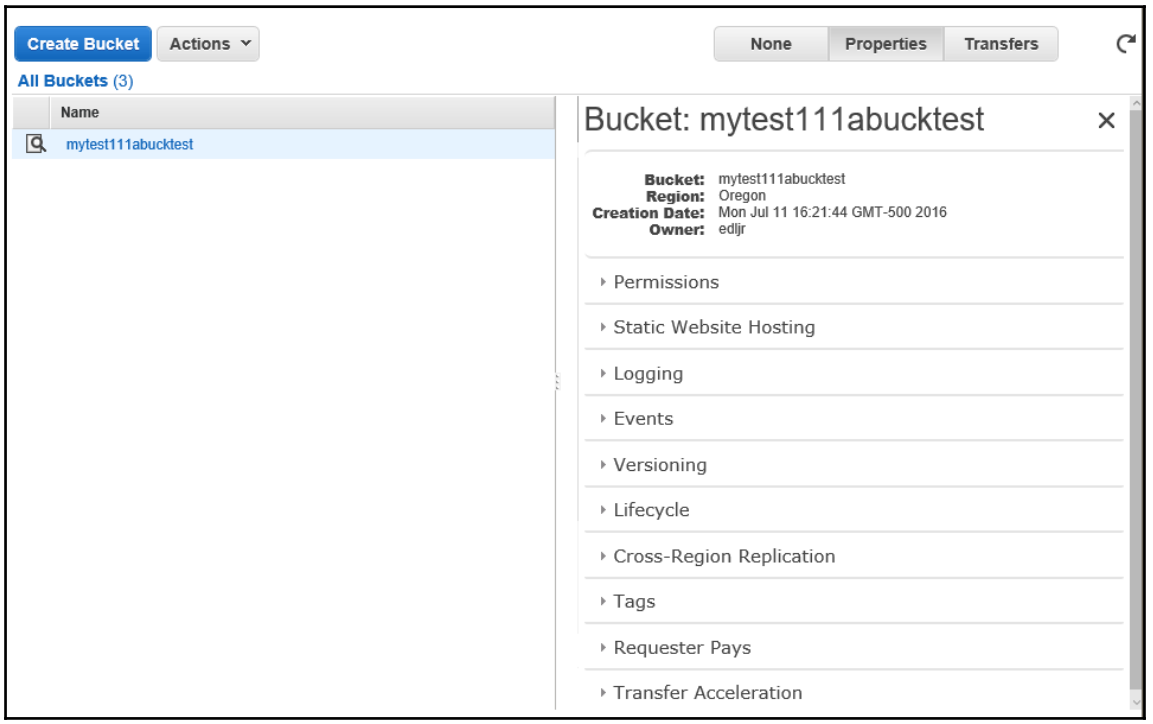
3. Click the **Create Bucket** button located in the top-left corner of the screen. Amazon S3 uses buckets as data repositories or containers.



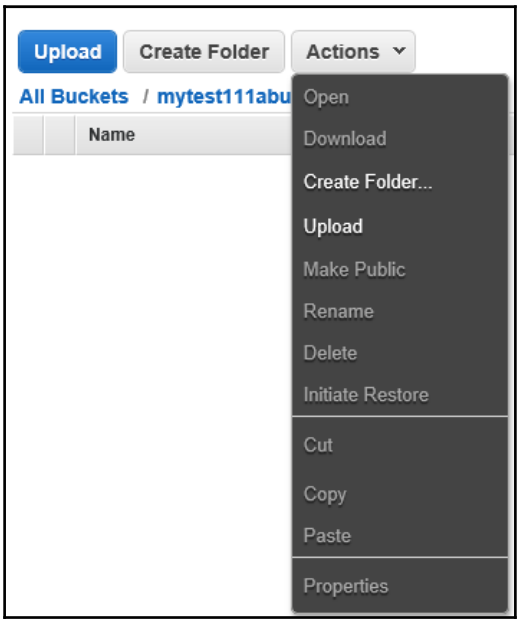
4. Click the **S3** link to open the S3 Management Console.
5. Create a bucket name, select a region, and click the **Create** button. The system will check to ensure your bucket name is unique. If you entered a non-unique bucket name, you will be prompted to enter a new name:



6. You will now see the newly created bucket in the left column and its properties listed in the right column. You might need to select the **Properties** button at the upper-right in order to see the properties listed:



7. Click on the bucket you just created. This will result in a file-folder type of interface for your bucket. Here you can upload objects, create a folder structure, and perform additional bucket operations through the **Actions** pull-down menu:



When you start adding objects to your buckets, you will incur charges. There is no cost for creating buckets, just for housing the data you fill them with.

Amazon S3 API

In the previous section, we created an S3 bucket and illustrated how to manually create folder structures and to upload data objects. In this section, we will review the Amazon S3 API so that you can programmatically manage your buckets.

It is important to note that Amazon S3 is not specific to Lumberyard. In fact, it is programming language-agnostic, which means you can access your buckets through the Amazon S3 API regardless of the programming language you are using.

Like other cloud storage solutions, you interface with Amazon S3 using the **Representational State Transfer** (**REST**) **API** for HTTP and **Simple Object Access Protocol** (**SOAP)** for HTTPS.

Details regarding REST, SOAP, and the Amazon S3 API are beyond the scope of this chapter and can be freely explored at the following AWS web page: `http://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html#API`.

Summary

In this chapter, we reinforced our understanding of the need for cloud-based solutions for the games we develop in Lumberyard. While this requirement is not unique to Lumberyard, it is uniquely easy to integrate cloud-based solutions using Lumberyard and AWS. We reviewed how to enable the Cloud Canvas gem in our game. We also created credentials and created an IAM user. You gained experience with the Canvas Resource Manager. In addition, you gained insight into the Amazon S3 and used it to create a data bucket. We concluded the chapter with a brief discussion on the Amazon S3 API, REST, and SOAP.

In the next chapter, you will be introduced to Twitch, the AWS that allows people to watch live game streaming. You will also learn how to implement Twitch functionality for in-game user interactions. Key concepts will include *Twitch JoinIn* and *Twitch ChatPlay*.

10
Engaging With Users Using Twitch

In the previous chapter, we explored AWS with a specific look at Cloud Canvas and Amazon S3 for cloud computing and storage respectively. Our exploration included an overview of the need for cloud-based solutions to enhance Lumberyard games. Using a hands-on approach, we used the AWS Console to create an **Identity and Access Management** (**IAM**) user. We also employed the Cloud Canvas Gem and learned how to access Amazon S3.

In this chapter, we will take a singular look at Twitch, the Amazon Web Service that allows people to watch live game streaming. You will also learn how to implement Twitch functionality for in-game user interactions. Key concepts will include Twitch JoinIn and Twitch ChatPlay.

After reading this chapter, you will:

- Understand the concept of video game broadcasting
- Understand the components of Twitch
- Have your own Twitch account and **Broadcast Channel**
- Be able to instantiate Twitch ChatPlay in a Lumberyard game
- Be able to use Twitch JoinIn to invite viewers
- Understand the basic constructs of the Twitch API

Don't jerk, Twitch!

A twitch is defined as a sudden movement or jerk. The Twitch that we care about is a noun, not a verb. Twitch launched in 2011 as an online video gaming broadcast network. The online service joins gamers and game enthusiasts for live and recorded events. Uses for the service include broadcasting your games, watching live game streams, watching recorded videos of previous gameplay, and engaging in chat. The site's monumental growth in popularity is one of the aspects that led Amazon to purchase Twitch. This is a bonus for Lumberyard developers because Twitch and Lumberyard are compatible.

Video games are often played with some sort of social component. A classic example of this is playing a first person shooter console game with headsets that allow you to communicate with your team. Many online games support in-game text chat. While some might see this as a distraction, the industry has seen social components of games steadily increasing. Twitch takes this to another level, allowing fan bases to watch live gameplay. More than that, they can chat during the game play. This has given rise to tournaments and a validated place for services such as Twitch in the game space.

Twitch humor: So, where did the name Twitch come from? One must wonder if the creators of Twitch were catering to the idea that excessive gameplay can cause involuntary muscle twitching. We might never know for sure.

Dissecting Twitch

It might seem like integrating Twitch in our Lumberyard games is a simplistic task. After all, we simply want to enable Twitch or not enable Twitch, right? There is a lot that goes on behind the scenes that we, as game developers, must contend with to enable Twitch in our games.

In this section, we will consider what is possible with Twitch and look at the components necessary to fully integrate with Twitch: *Twitch ChatPlay*, *Twitch JoinIn*, and the *Twitch API*.

You can think of Twitch in two ways. First, there is the service that is accessible at `https://www.twitch.tv/`. Although this service is owned by Amazon, it is where the end result of our efforts lies and is therefore not our focus. The second way to think of Twitch is as a set of tools and APIs to enable real-time socialization of our Lumberyard games.

What is possible with Twitch?

By now you know that we can set our games to stream on Twitch and enable chat. What are the capabilities of Twitch besides streaming games? What specifically are its capabilities? These are great questions. Here is a list of supported functionalities:

- Creating custom chat commands
- Viewer polls for voting
- Viewer surveys
    - Inviting targeted viewers to game sessions

Let's consider why we might want to use each of these features.

Creating custom chat commands

We design our games to be fun and our success in that endeavor is directly linked to the game's overall success. We can take the same approach to in-game chat, as well as the chat system on Twitch. You can create game-relevant custom chats that, when entered by users, impact in-game behavior. For example, you might create a `#godeep` chat command that causes AI creatures to burrow in the ground. You are only limited by your imagination.

Viewer polls and surveys

Twitch ChatPlay Voting allows you to create polls and surveys for voting. How you implement the polling or survey results in your game is something to consider during design time. Alternatively, you can create polls and/or surveys as post-game feedback mechanisms.

Inviting targeted viewers to game sessions

When your game is being played and broadcast to Twitch, the person playing the game is considered a Twitch broadcaster. Using Amazon GameLift and Twitch JoinIn, broadcasters have the ability to target and invite game viewers. You learned about Amazon GameLift in `Chapter 7`, *Creating Multiplayer Gameplay*. We will discuss Twitch JoinIn later in this chapter.

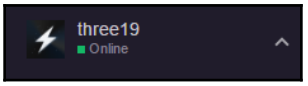
Creating a Twitch Channel

One of the requirements for using Lumberyard and Twitch is to have an active Twitch account and **Channel**. Accounts are free and easy to create. The following steps will walk you through the process:

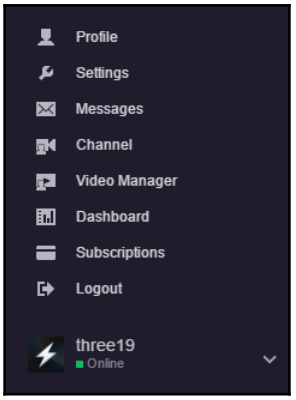
1. Point your browser to `https://www.twitch.tv/`.
2. Click the **Sign Up** link located in the top-right corner of the website.
3. Fill in the form to create your account and click the **Sign Up** button at the bottom of the **Sign Up** form:

4. Check your e-mail account and click the link to verify your e-mail address. If you do not do this, you will see a reminder in the top banner of the Twitch site when you log in:

5. Log in to your Twitch account and go to your channel.
6. In the lower-left corner of the site, you will see your account tab. Click the up arrow to the right of that tab to expand your settings:



6. You will now have access to your settings. Click the **Channel** link:



7. With your channel displayed in the main view section of the Twitch website, click the **Edit** link and change the title of your broadcast to something unique:

Now that you have created your Twitch account and established your broadcast channel, you are ready to create a game in Lumberyard implementing Twitch.

Implementing the Twitch ChatPlay system

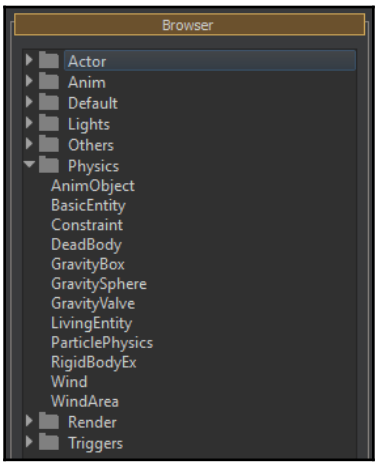
Twitch ChatPlay is a framework used by Lumberyard to implement Twitch connectivity and functionality. To demonstrate its use, we will first create a new game level. We will perform minor edits on the game level to include some terrain and the addition of 3D objects. Next, we will incorporate Twitch ChatPlay into our level. Finally, we will test our game. The steps to complete each of these three objectives are presented in the next three subsections.

Objective 1 – Creating a new game level

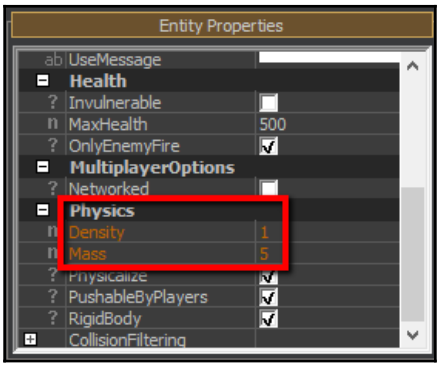
Before we can implement Twitch ChatPlay, we need a game to use. We could use a sample game level, such as one of those we have worked with in previous chapters. Instead, we will create a simple game level as the basis for our testing. Here are the steps:

1. Launch the **Lumberyard Editor**.
2. Click the **New level** button.
3. In the **New level** dialog window, enter a name for your level, such as `TwitchTest`.
4. Click the **OK** button on the **New Level** dialog window.
5. The **Generate Terrain Texture** dialog window will appear next. You can accept all of the defaults and click the **OK** button.
6. Our level is now created, so we are ready to edit the terrain. In the **RollupBar**, select the **Terrain** tab.
7. Click the **Environment** button to display the **Environment Properties**.
8. Under **EnvState**, change the **Wind vector** property setting from `1,0,0` to `0,0,0`. This will essentially disable the wind on the level. We are disabling the wind to make our testing much easier.
9. Now we are ready to create some 3D objects for our game. In the **RollupBar**, select the **Create** tab. This displays the **Objects** section.
10. Click the **Entity** tab. This will display a hierarchical browser.

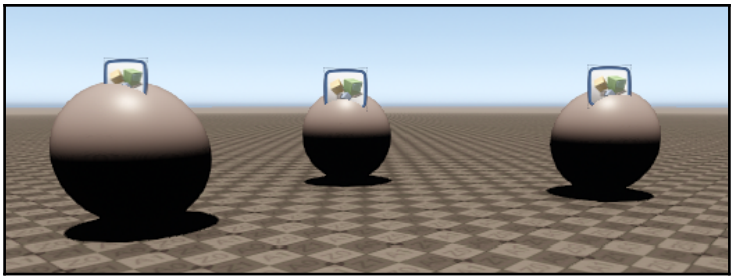
11. In the hierarchical browser, expand the **Physics** entry. This will display the various physics objects available to you:



12. Click **BasicEntity** in the hierarchical list, drag it to the game level, and click to place it.
13. We are going to animate this object using Twitch ChatPlay, so we need to change a few settings. In the **Entity Properties** dialog, under **Physics**, change the **Density** to `1` and the **Mass** to `5`:



14. Repeat steps *12* and *13* twice, so you have a total of three orbs in your level:

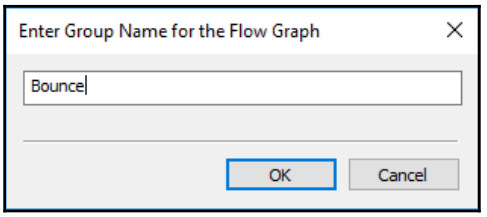


Our game level is now set up. We have three identical orbs. In the next section, we will use FlowGraph to integrate Twitch ChatPlay in our game level.

Objective 2 – Twitch ChatPlay integration

In the previous section, we created a new game level with three orbs. In this section, we will create Flow Graphs for each orb and assign unique behaviors that will be instantiated in Twitch using ChatPlay. Here are the steps for the first orb:

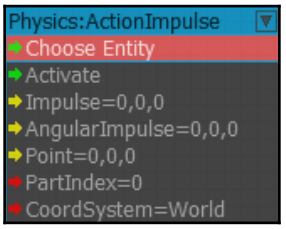
1. Right-click your first orb and select **Create Flow Graph**.
2. In the **Enter Group Name** for the **Flow Graph** dialog window, enter `Bounce` as the name and click the **OK** button. This will bring up the **Flow Graph Editor**:



We are using a **Flow Graph** group so the Flow Graphs for all three of our orbs will be nicely organized in the **Flow Graph Editor**.

3. In the **Components** pane in the left section of the **Flow Graph Editor**, expand the **Physics** section to expose the node types.

4. Drag an **ActionImpulse** node to the Flow Graph. Now we have a **Physics:ActionImpulse** node in our graph:



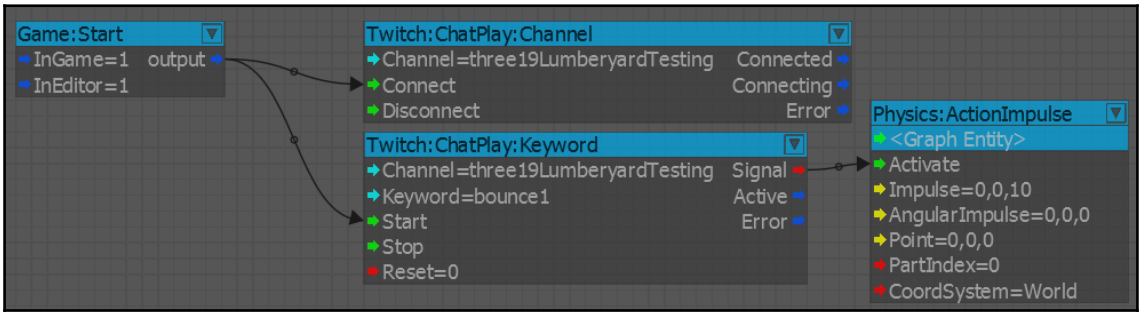
5. Right-click **Choose Entity** and select **Assign** graph entity.
6. Next we will change the **Impulse** setting to simulate a small bounce action. Change the **Impulse** values from `0,0,0` to `0,0,10`.
7. In the **Components** pane in the left section of the **Flow Graph Editor**, expand the **Game** section to expose the node types.
8. Drag a **Start** node to the Flow Graph.
9. In the **Components** pane on the left section of the **Flow Graph Editor**, expand the **Twitch** section to expose the node categories. You will see here that the **Twitch Flow Graph** node categories are **API**, **ChatPlay**, and **JoinIn**.
10. Expand the **ChatPlay** section to reveal the available nodes.
11. Drag a **Channel** node to the Flow Graph.
12. Edit the **Channel** input of the **Channel** node to reflect your Twitch Channel name.

Twitch Channel names are not always the same as a broadcast name. Be sure you know what names you have assigned.

13. Drag a **Keyword** node to the **Flow Graph**.
14. Edit the **Channel** input of the **Keyword** node to reflect your Twitch Channel name.
15. Edit the **Keyword** input of the **Keyword** node and enter `bounce1`. This will be the text that users can enter in your Twitch channel's chat to cause your first orb to bounce.

16. Now it is time to connect the nodes. Drag **Game:Start output** to **Twitch:ChatPlay:Channel Connect input**.

17. Drag **Game:Start output** to **Twitch:ChatPlay:Keyword Start input**.
18. Our last connection involves dragging **Twitch:ChatPlay:Keyword Signal output** to **Physics:ActionImpulse Activate input**.
19. After following the preceding steps, your Flow Graph should look similar to the following screenshot. Your channel name will be different, but the rest should be the same:



20. You can now optionally replicate the steps above to create Flow Graphs for the remaining two orbs. For additional testing, you can assign `bounce2` and `bounce3` as the keywords and give them different *Z* values in the **ActionImpulse** node.
21. When you create the additional Flow Graphs, be sure to assign the new Flow Graphs to the **Bounce Group** we created in step 2 previously.
22. Here is a suggested set of values for the new Flow Graphs:

Orb	Keyword	Impulse Value
1	`bounce1`	`0,0,10`
2	`bounce2`	`0,0,25`
3	`bounce3`	`0,0,50`

Objective 3 – Testing

Now we are ready to test our game level. We'll cover the steps required to test the game level and conclude with some troubleshooting tips.

Follow these steps to test your game level:

1. Close the **Flow Graph Editor**.
2. Save your level.
3. Enter Game Mode. You can accomplish this with the *Ctrl* + *G* keyboard combination or through the **Game** | **Switch to Game** menu selection.
4. With your game in Game Mode, go to your browser and open your channel in Twitch.
5. In the Twitch channel chat, type `bounce1` and hit the **Chat** button.

Pressing the *Enter* key on the keyboard is the same as clicking the **Chat** button. If you are using a mobile device to connect to Twitch (yes, there's an app for that), then you'll type `bounce1` and hit the **Send** button.

6. You will now see your first orb's *Z* axis increase from `0` to `10`, then the orb will bounce and roll to a stop. You can do this repeatedly.
7. If you created Flow Graphs for the other two orbs, you also can test the `bounce2` and `bounce3` keywords.

The keywords are not case-sensitive, so typing `Bounce1` and `bounce1` in the chat will result in the same behavior.

As you can see, testing our connectivity between our Lumberyard game level and Twitch is pretty straightforward. If this did not work for you, review the following troubleshooting tips:

Possible Problem	**Resolution**
Cannot switch focus from the game to browser.	Before putting your game in Game Mode, open your browser and bring up your Twitch Channel. Once the game is in Game Mode, use the *Alt* + *Tab* keyboard combination to navigate to your browser. You may need to switch back to the game to see the results.
Nothing happens when I enter the keyword in chat.	Check to ensure your channel names are identical in both the **Twitch:ChatPlay:Channel** and **Twitch:ChatPlay:Keyword** nodes and that they match the name of the channel in Twitch. *Spelling matters!*
I verified the channel names and nothing happens when I enter the keyword in chat.	There are two things to check here. First, check that keywords are correct. Although case does not matter, spelling does. Ensure what you are typing in the chat matches what you entered in the **Twitch:ChatPlay:Keyword**node, in the **Keyword** field. Secondly, check your Flow Graph nodes to ensure they match the three connections we created earlier in this chapter.

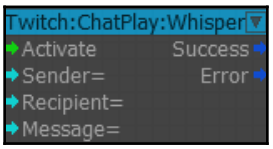
Understanding Twitch JoinIn

As you will recall from earlier in this chapter, when you integrate Twitch into your game you are considered a **Twitch Broadcaster**. You are broadcasting your game to your Twitch Channel for others to watch and, if so enabled, use Twitch ChatPlay to interact with your game. So, how do people know about your game on Twitch? How you advertise and announce your channel is up to you. If you want to target specific viewers in your game, you can use *Twitch JoinIn*.

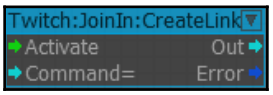
Twitch JoinIn uses GameLift session information and a Flow Graph node to provide linkages between users and your game. We covered GameLift in `Chapter 7`, *Creating Multiplayer Gameplay*, and will discuss the appropriate Flow Graph node in this section.

Twitch JoinIn uses the **Twitch:ChatPlay:Whisper** and **Twitch:JoinIn:CreateLinkFlow Graph** nodes.

The **Twitch:ChatPlay:Whisper** node, shown here, is used to send game session information to your targeted viewers. The session information is sent via a link that can be clicked on viewer computers to launch the game.



The **Twitch:JoinIn:CreateLink** Flow Graph node, shown here, has two inputs and two outputs. When the **Activate** input is signaled, a protocol link is generated. The **Commands** input can be used to pass specific commands for the game's initial launch. There is an **Out** output that outputs the aforementioned protocol link and an **Error** output to allow you to perform exception handling.



Twitching with the Twitch API

An API, or Application Programming Interface, is a set of software tools for creating applications that integrate with other software or services. In our context, the Twitch API defines how we can programmatically integrate our game with Twitch.

In Lumberyard, we can use the **Twitch:API:GET** Flow Graph node to make calls to the Twitch API. That node, shown below, has three inputs and two outputs. The inputs consist of the channel name, the specific **API_Key**, and **Get** to start the API call. The outputs are used for returned values and error handling.

Using the API, we can go beyond what is available to us using Lumberyard's **Flow Graph Editor**. The Twitch API and supporting documentation are available at `https://github.com/justintv/Twitch-API`.

Summary

In this chapter, we explored the use of Twitch and how to integrate Twitch with the games we create in Lumberyard. We started with a discussion on video game broadcasting and then explored Twitch as a service. We walked through the steps to create a Twitch account and a Broadcast Channel. Using a blank game level, we created 3D objects and used the **Flow Graph Editor** to instantiate Twitch ChatPlay. Our efforts resulted in a game that could be interacted with live chat on Twitch. We concluded our Twitch coverage with an overview of Twitch JoinIn and the Twitch API.

In the next chapter, we will look at how to wrap up your game and prepare it for distribution. We will explore several types of builds, including game builds, testing builds, debugging builds, and release builds. We will discuss methods of debugging, testing, releasing, and supporting your games.

11
Providing Your Game to the World

In the previous chapter, we explored Twitch, Twitch ChatPlay, Twitch JoinIn, and the Twitch API. We learned how the use of Twitch ChatPlay allows game viewers to directly interact with games we create in Lumberyard. We demonstrated this feature in the previous chapter, giving you hands-on experience at creating that type of interaction.

The aim of this chapter is to provide an overview of the steps necessary to publish your game once it is completed, that is, make your game available for others to play. Specifically, we will look at game builds, and how to test them, debug them, and release them.

After reading this chapter, you will:

- Learn which distribution platforms Lumberyard supports
- Grasp the requirements for publishing to gaming consoles
- Understand the complexities involved in generating game builds
- Be able to identify the three types of game build supported by Lumberyard
- Learn the components and requirements of each game build type
- Understand special considerations when working with a beta game engine
- Gain exposure to Lumberyard's testing tools

Taking your game beyond the Lumberyard Editor

So far we have created game levels and explored existing sample game levels to learn how to use the Lumberyard Editor and its primary functionality. In each case, we played the game using Game Mode within the **Editor**. That is great for testing your games, but you cannot expect users to have a copy of the **Lumberyard Editor**. So, we need to publish the game in a format that permits others to play it.

We can use Lumberyard to create games for computers running the Windows operating system and the Xbox One and PlayStation 4 game consoles. At the time of this book's publication, support for deploying to mobile devices running iOS and Android was in preview, and therefore not covered in this chapter. In addition, support for computers running the Mac and Linux will be forthcoming.

Next, we will look at the external requirements for publishing to the Xbox One and PlayStation 4 game consoles. Then, we will conclude this section with an overview of publishing to Windows-based computers.

Xbox One

Xbox One is a Microsoft product. Microsoft has a program called `ID@Xbox` – the Independent Developer Program at Xbox. This program allows developers to self-publish their games on Xbox One. In order to publish games for Xbox One, you must first register with Microsoft. Here is the link to get started:

```
http://www.xbox.com/en-us/Developers/id
```

It is recommended that you use Xbox One dev mode to test your games on your own device. You can do this without registering with Microsoft, but that registration grants you access to SDKs and additional support. And, to distribute your games, you must be registered.

Get yourself registered with Microsoft for free. Currently, when this book was published, there was no cost for registering with `ID@Xbox`.

If Microsoft approves your request to be a licensed Microsoft Xbox developer, you will need to provide your credentials to Amazon. You merely need to send an e-mail to `lumberyard-consoles@amazon.com`. In the e-mail, include your personal name, the name of your game studio (should match what is on your employer tax ID), and your e-mail address.

PlayStation 4

Sony, the company behind PlayStation 4, has a program where developers can become *PlayStation Partners*. Once you are a PlayStation Partner, you are able to self-publish titles on the PlayStation network.

You will need to have an employer tax ID and proof of your corporate entity. Also, you must reside in one of the following geographic areas: United States, Canada, Mexico, Central America, or South America. Review the PlayStation developer site (`https://www.playstation.com/en-us/develop/`) for the latest requirements.

To apply to become a partner, developers first submit an online application. This registers your game studio with Sony and gets your application submitted. Once approved, you will need to sign an agreement with Sony. Then, you will be an official PlayStation Partner.

Become a PlayStation Partner while it is free. At the time this book was published, there was no cost for registering with the PlayStation Network.

Once you are a licensed Sony PlayStation Partner, visit the Sony Computer Entertainment Developer Network at `http://www.scedev.net/` and navigate to the **Middleware Directory**. Once there, click the **Confirm Status** link for Amazon Lumberyard.

You may not be able to see the **Middleware Directory** until you are registered with SCE and are logged in.

Publishing to Windows-based computers

Publishing to Windows-based computers is a lot less restrictive than publishing titles on game consoles. Essentially, there are no restrictions.

One thing to consider is how you will host your game. If you are creating a standalone, non-network game, then this is not an issue for you. As you will remember, Amazon Lumberyard is a game development suite that is primarily used to create AAA games. These games are typically hosted on a server or set of servers. You can host your Lumberyard games on your own servers or with any hosting provider (such as Steam). Hosting your games with Amazon, using the AWS, is most likely the easiest and most cost-effective solution. When the time is right, you can review your options and make a final decision.

Generating game builds

When you complete your game, it will likely consist of several hundred files. These files will include libraries, scripts, textures, object files, and more. Creating a game build is the process of combining these files into a compiled version of your game. Lumberyard supports the creation of several types of build.

We can create *release builds* that can be used to distribute our game. The same type of build would be used for demo versions of a game. This type of release compiles the game into an `*.exe` file. So the build will consist of a single file, such as `yourgame.exe`.

Lumberyard also supports generating a *debug game build*. It is used to debug the game and includes several tests including thorough memory tests.

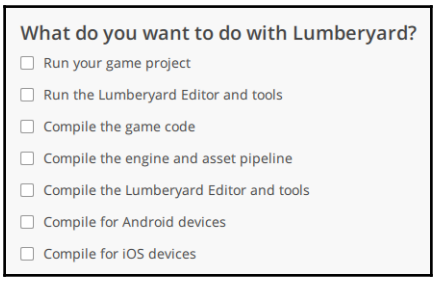
The third type of game build in Lumberyard is the *profile build*. This type of build is ideal for performance testing.

Release builds

Release builds are generated in order to provide your game to others in a playable format. This section provides an overview of how to generate a release build.

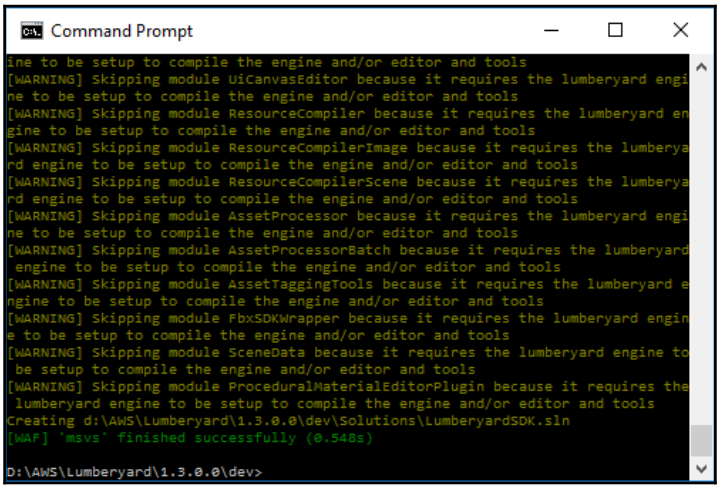
The release process is quite complex and involves using the `Waf Build System`. The remainder of this section provides an overview of the process. Please consult the Lumberyard documentation (`http://docs.aws.amazon.com/lumberyard/latest/userguide/waf-intro.html`) for details on the build process. The `Waf Build System` is specifically covered in `Chapter 12`, *Stretching Your Lumberyard Wings*.

We can compile our game code using one of two different methods. When we launch the **Lumberyard Setup Assistant** and click **Get Started**, we are provided with seven different options. Up to this point, we have only used the **Run the Lumberyard Editor and tools** option.



When the **Compile the game code** option is selected, we can click **Summary** in the left navigation pane, then click the **Configure project** button. This takes you to the **Project Configurator**. Here you will select the appropriate project and close the **Project Configurator**.

At this point, you have two steps remaining. First, you will run the `lmb_waf` batch utility with the configure parameter. Do this by typing the following on a command line in a Command Prompt window: `lmbr_waf configure` and pressing the *Enter* button. This process can take a while and will result in a lot of information being scrolled on your screen. When the process completes, you will receive a `[WAF] 'msvs finished successfully` message.



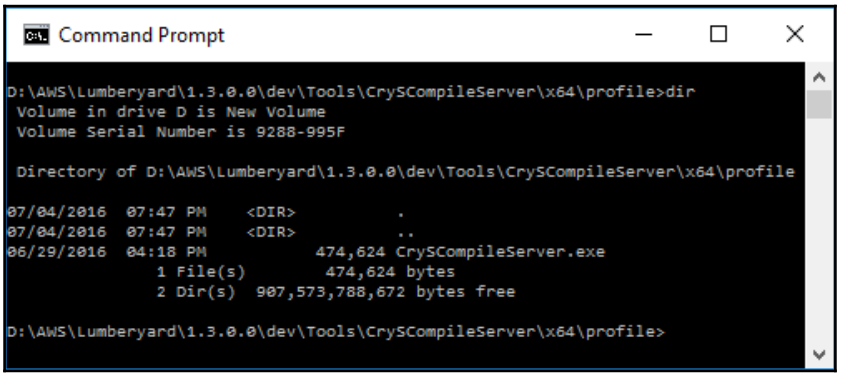
Review the output closely. You will need to address all reported errors.

For the compile process to work, you will need Python 2.6, or a later version, installed on your computer. Here is the download site: `https://www.python.org/downloads`.

Your last step is to run the `lmbr_waf batch utility` again, but with a new set of parameters. You will enter the following on the command line in a Command Prompt window: `lmbr_waf build_win_x64_profile - p game`, and then press the *Enter* key.

Now you have the code compiled. You also need to compile your game's assets. Lumberyard provides three tools for compiling assets into `.pak` files:

- The Shader compiler tool is located deep in the Lumberyard file structure on your computer and generates a `ShaderList.text` file. The following screenshot illustrates where the Shader compiler, `CrySCompileServer.exe`, is located:



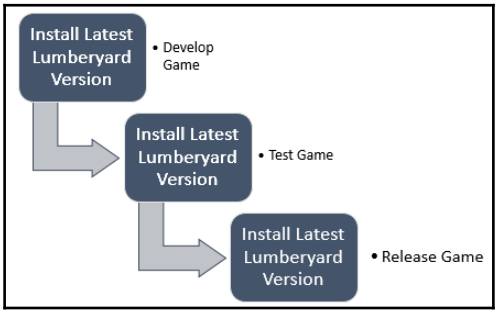
- `ShaderCacheGen.exe`, located in the `\dev\Bin64` folder, takes the `ShaderList.txt` file and creates a folder containing all the assets on that list.
- `BuildShaderPak_DX11.bat`, located in the `\dev folder`, generates the `.pak` files.

With the code and assets compiled, follow the steps outlined next to generate your release build:

1. Open a command prompt window.
2. Change to your `\dev` directory (that is, `D:\AWS\Lumberyard\1.3.0.0\dev`).
3. Enter `lmbr_waf build_win_x64_profile -p game_and_engine` on the command line and then press the *Enter* key.
4. Next, enter `BuildSamplesProject_Paks_PC` on the command line and press the *Enter* key.
5. Move or copy all of the `.pak` files so they are in the same directory.
6. Copy (not move) the `\GameLift\GameLiftSampleClient\Bin64.Release` directory to the directory you created in step *5*.
7. Now you can run the game executable file from the `Bin64.Release` folder you created in step *6*.

Every game is a bit different and the process of generating a release build is complex. The steps provided here are intended to provide you with an overview of the process. There are no release build-related concerns that you need to keep in mind when designing or developing your game. When your game is ready for release, you should consult the Lumberyard documentation for the latest information regarding release builds.

Since, at the time of this book's publication, the Lumberyard game engine was still in beta, it is advisable to ensure you have the latest engine version installed on your computer prior to publishing a release. It is likely that the game engine will have changed during the period from when you started your game project until the time you are ready to publish it. That being said, it is a good idea to test your game on the latest version of the game engine prior to release. The following graphic provides a suggested workflow regarding beta game engines:



As a final note about *release builds*, it is important to remember the following information and rules about the process:

- The build process is not a single-step operation. So, compiling assets (that is, `shaders`) and code is a necessary prerequisite, as indicated in the previously detailed steps.
- Lumberyard's Asset Processor is not part of the release build and is therefore not available at runtime.
- None of the testing or performance tools associated with the *debug and profile build* types are included. Those build types are discussed in the next section.
- The end result will be a single executable (`*.exe`) file. There are no external libraries or repositories.

Debug and profile builds

In the previous section, we looked at providing release builds so people outside your team can play your game without needing **Lumberyard Editor**, the Asset Processor, or other Lumberyard components. Release builds can be considered the final type of build. There are additional build types, including debug and profile:

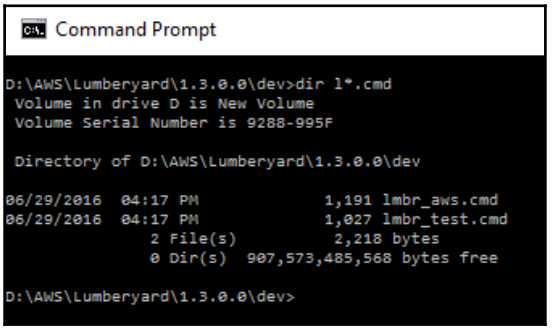
- Profile builds can be generated to allow team members, or extended team members, to perform some debugging and testing. There are several tools, discussed in the next section, associated with this type of build, giving developers great insight into the efficiency of their game. Unlike release builds, profile builds require Lumberyard, the Asset Processor, and other Lumberyard utilities as runtime. So, this type of release is typically useful to people on your game project team.
- Generally, the steps provided in the earlier release build section can be followed. Step *3* should be replaced by entering the following on the command line in a command prompt window: `lmbr_waf build_win_x64_profile – p all`, and then pressing the *Enter* button.
- Debug builds are intended for, as you would imagine, debugging. These builds are *rough* versions of profile builds that have additional tests and memory checks.

Lumberyard's testing tools

In the previous section, we indicated that debug and profile builds can be used for debugging and testing your game. Debugging and testing are usually internal to a game project's team. Lumberyard supports these efforts with a couple of utilities. In this section, we will provide an overview of two tools that are part of Lumberyard's utility set: `AzTestScanner` and `Statoscope Profiler`.

AzTestScanner

The `AzTestScanner` utility can be run manually from the command prompt. You will need to locate the `AZtestRunner.exe` file on your system. Alternatively, you can use the `lmbr_test.cmd` script. This script file, located in the `\dev` directory, sets local and Python paths, sets a few properties, and then runs the command `python -m aztest`.



This scanning utility tests libraries and generates three reports. The first report is a detailed text log of the scanning results. Each library and executable file tested will have a corresponding XML file generated with test results specific to the library or file tested. Lastly, an HTML file will be generated and contains a summary of the entire scan's test results.

To fully use the `AzTestScanner` utility, you will need Python 2.6, or a later version, installed on your computer.

There are optional parameters that can be used to tailor the scans. Consult the documentation (`http://docs.aws.amazon.com/lumberyard/latest/userguide/testing-aztestscanner.html`

) for details.

Statoscope Profiler

The `Statoscope Profiler` is an advanced tool that provides you with key information regarding your game's performance, memory usage, and additional statistics. This tool is located in the `\dev\Tools\Statoscope` directory.

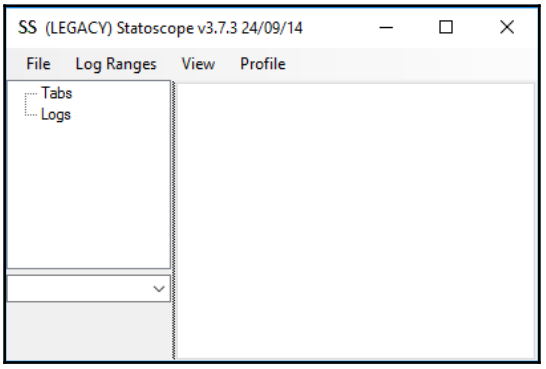
When launched without a connected game, the interface is, as shown here, simplistic. The menu options are essentially self-explanatory and are listed:

File: **Open Log**, **Save Log**, **Connect**, **Exit**

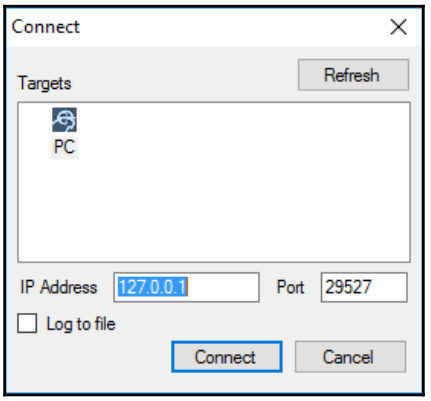
Log Ranges: **Reset**, **Crop to View**

View: **Graph**, **Fit to Frame Records**, **Screenshot Graph**

Profile: **Open**, **Save**



The tool needs to be connected to a game. Once a running game is connected, the graph will start to populate and allow you to monitor performance over time:



Summary

In this chapter, we reviewed, at a high-level, the steps necessary to publish your Lumberyard games so that they are accessible to others. We looked at the different distribution platforms supported by Lumberyard and additional requirements for console development. We reviewed the three types of game build (release, debug, profile) and their uses. We also looked at two of the testing tools bundled with Lumberyard.

In the next (and final) chapter, we will take a glimpse at what is possible with Lumberyard, beyond the basics. We will explore concepts such as Virtual Reality, the Waf Build system, Lumberyard's cinematics system, system streaming, and memory handling. We will also explore Amazon Web Services to uncover key additional services not covered in previous chapters.

12
Stretching Your Lumberyard Wings

In the previous chapter, we reviewed, at a high level, the steps necessary to publish your Lumberyard games so that they are accessible to others. We looked at the different distribution platforms supported by Lumberyard and additional requirements for console development. We reviewed the three types of game build (release, debug, profile) and their uses. We also looked at two of the testing tools that come bundled with Lumberyard.

In this (final) chapter, we will take a glimpse at what is possible with Lumberyard, beyond the basics. We will explore concepts such as VR, the Waf Build system, Lumberyard's cinematics system, system streaming, and memory handling. We will also explore two additional Amazon Web Services (**Simple Query Service** and **Simple Notification Service**) to round out our coverage of Amazon Web Services throughout this book.

In this chapter, you will:

- Learn how Lumberyard supports VR
- Understand details of the Waf Build system
- Appreciate the capabilities of Lumberyard's cinematics system
- Gain exposure to the basics of system streaming
- Understand the importance of memory handling for your Lumberyard games
- Learn the basics of the Simple Queue Service
- Learn the basics of the Simple Notification Service

Virtual Reality and Augmented Reality

Virtual Reality (**VR**) and **Augmented Reality** (**AR**) are two of the most exciting trends in game development. The two technologies are similar. Virtual Reality is when you create an environment in which the user can interact as if it were real. Augmented reality is when our current reality is augmented with something virtual. Here are examples:

- **VR**: A user dons a headset and receives audio and video from a virtual game. The user sees, through this headset, a world that seems real. They have no sensory input from the actual world. They might be on a spaceship or in the Amazon jungle. They are not watching this on a television, they are experiencing it through Virtual Reality hardware.
- **AR**: Imagine holding up your phone and viewing the screen based on your camera's video input. In this scenario, you simply are seeing what is in front of you. An augmented reality application might have animals or other creatures roaming the actual terrain around you.

We looked at both virtual and augmented reality so the difference is clear. Let's now look at how Lumberyard supports Virtual Reality.

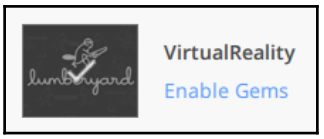
VR hardware

There is a growing number of hardware devices that support VR applications. These devices are head-mounted. Lumberyard specifically supports Oculus Rift (`https://www.oculus.com/rift/`) and HTC Vive (`http://www.htcvive.com`). You can also configure your game for other Virtual Reality head-mounted devices.

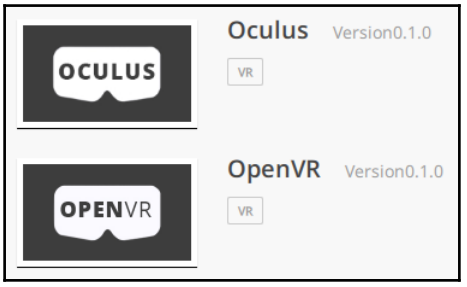
Setting up your VR project

Lumberyard has two *Gems* that support Virtual Reality: *Oculus* and *OpenVR*. The steps to enable these *Gems* are as follows:

1. Launch the Lumberyard**Project Configurator**.
2. Click the **Create New** button and select a name for your project, such as `VirtualReality`.
3. Your project will now display with an **Enable Gems** link underneath the project name. Click that link.



4. Select the game Gem that corresponds to the Virtual Reality head-mounted device you are targeting. Select **Oculus** for Oculus Rift and **OpenVR** for HTC Vive. You can also select both Gems.

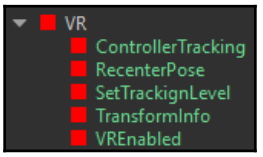


5. Click the **Save** button.
6. Click the **Back to Projects** link.
7. Click the **Set as Default** button.

You will likely have to rebuild your project using the command prompt. Consult the Lumberyard documentation (`http://docs.aws.amazon.com/lumberyard/latest/userguide/gems-system-gems.html`) for specific commands and procedures for accomplishing this.

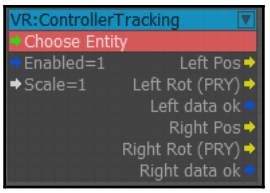
Flow Graph nodes that support VR

There are five Flow Graph nodes available to you:



Let's take a brief look at each one of the aforementioned Flow Graph nodes:

- **ControllerTracking**: VR games will utilize a head-mounted device as well as an input controller. Both devices provide inputs and it is important that they be properly aligned. You can use the **VR:ControllerTracking** Flow Graph node to align the VR head-mounted movement to the input controller.



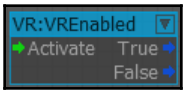
- **RecenterPose**: The **RecenterPose** Flow Graph node is helpful when you are designing a game that incorporates a centered graphic, such as how hands are used in first person shooter games. You might, for example make a call to this node each time a new weapon is picked up or changed. As you can see from the following screenshot, there is only one input, **Activate**.

- **SetTrackingLevel**: The **SetTrackingLevel** Flow Graph node allows you to set the tracking level of your VR device to either head or floor. This is used for frame calculations.

You probably noticed that the **VR:SetTrackingLevel** Flow Graph node has a typographical error in the title banner. This error is also evident in the hierarchical list of nodes in the **Flow Graph Editor**. Remember, at the time of publication, Lumberyard was still in beta, so there are likely to be more examples of this type of error.

- **TransformInfo**: The **TransformInfo** Flow Graph node provides your game with the position and rotation of the VR head-mounted device within the game.

- **VREnabled**: The **VREnabled** Flow Graph node is perhaps the simplest. It takes one input and then outputs a Boolean value of **True** or **False** to indicate whether the VR head-mounted device is connected and functioning properly.



Testing your VR games

You are already familiar with running games in Game Mode. This allows you to run your game from within the **Lumberyard Editor** for testing. When you have a Virtual Reality Gem enabled for your game, you will have an additional button on your bottom toolbar. When you click the **VR Preview** button, you will be able to preview your game using your own head-mounted VR device.

In addition to using the **VR Preview** button, you can test your VR game using the same method as non VR games. Simply select **Game | Switch to Game** from the top menu or use the *Ctrl + G* keyboard combination. To exit Game Mode, simply press the *Esc* key.

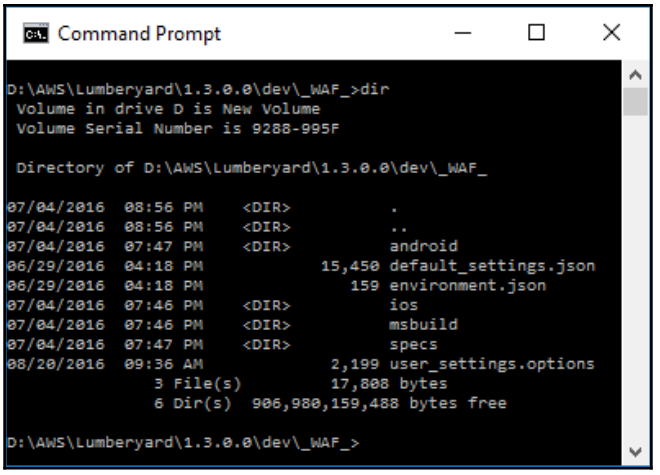
The Waf build system

You were first exposed to the Waf build system in `Chapter 11`, *Providing Your Game to the World* when you learned the requirements for releasing your Lumberyard games. *Waf* is Lumberyard's project build system. It can be used in conjunction with Visual Studio or you can use it manually from the command prompt.

You must have Python 2.6 or a later version installed on your computer to use the Waf build system.

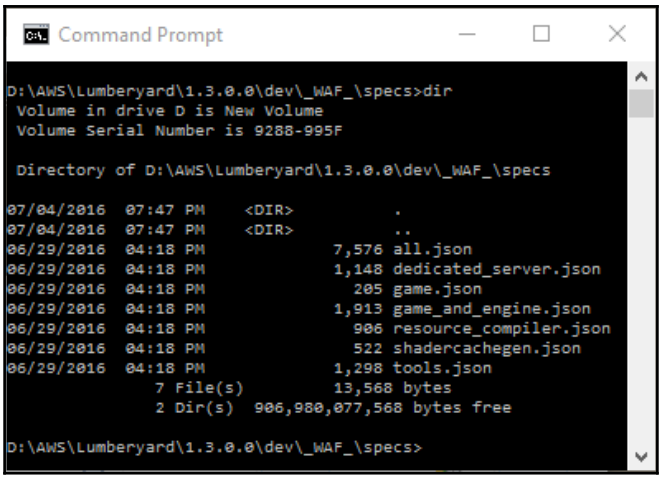
When you are creating simple, standalone games, such as single-player, non-networked games, you can simply use the **Project Configurator** as you have for sample projects in previous chapters. When you are creating a more complex game, you will need to get more involved in the process.

When you examine the Waf files structure, you will see a `_WAF_` folder in the Lumberyard `\dev` folder. As shown here, there are subfolders for Android, iOS, and Microsoft.



Within the Microsoft `Waf` folder, the **msbuild** sub-folder has a `waf build targets` file and an XML properties sheet. You can explore the contents of each file. They are both human-readable document formats.

Another folder worth reviewing is the `\dev\_WAF_\specs` folder. This is the folder that contains the Waf JSON files. JSON files are JavaScript Object Notation files and are used for data exchange. To learn about JSONs, you can visit the official JSON site at http://json.org for full documentation, or http://w3schools.com/json for tutorials. You can read more about this at:
`http://docs.aws.amazon.com/lumberyard/latest/userguide/waf-files-spec-file.html`.



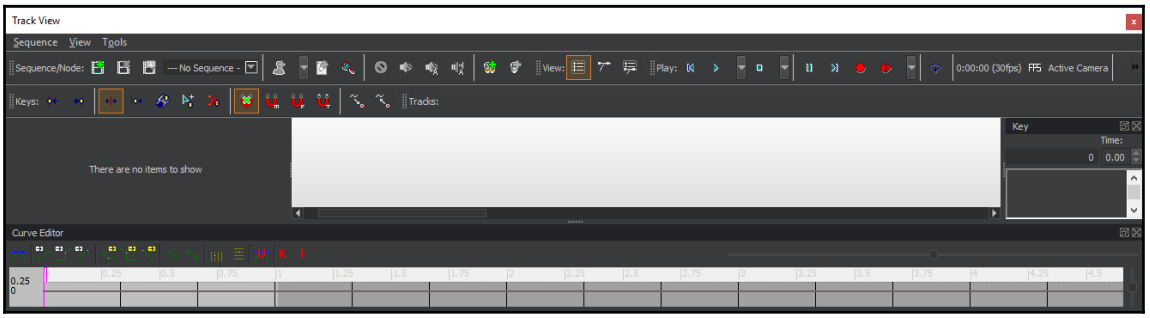
The Waf Build System is not unique to Lumberyard. You can visit the official Waf site at https://waf.io and the Waf GitHub site at `https://github.com/waf-project/waf`.

Lumberyard's cinematics system

Cinematics are a great way to showcase your game. They can be used as promotional videos and even as in-game cut scenes. These are more than simple animations; they are complex scripted events. Common uses are for reaching a boss level, death sequences, leveling up, and other grandiose game uses.

Most of the time, game cinematics are static. In Lumberyard, cinematics can be interactive. To access them, you select **View** | **Open View Pane** | **Track View** from the top menu. This opens the **Track View Editor**.

As you can see from the next screenshot, the **Track View Editor** is chock-full of functionality. Creating cinematics is a complex process that involves great levels of meticulous precision. The **Track View Editor** has an interface similar to animation and video editing software, which uses layers. Using the editor, you create scenes, layers, and sequences.

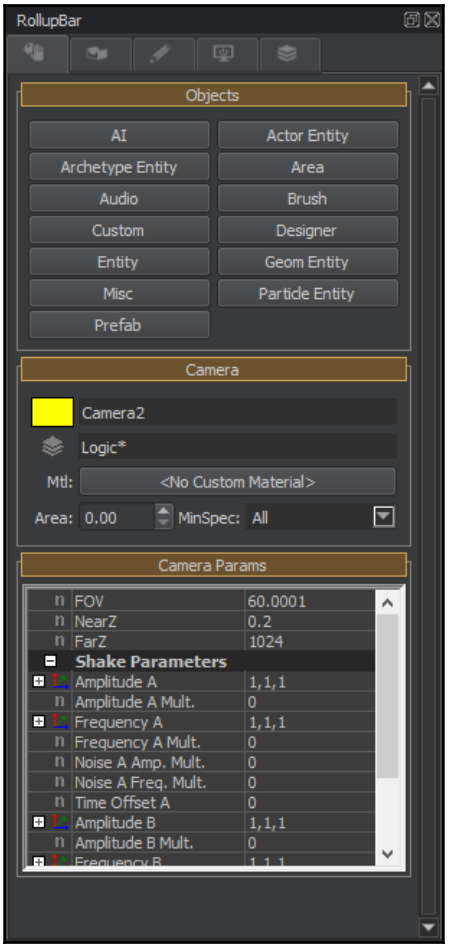


When working with cinematics in Lumberyard, it is helpful, as far as possible, to use previously created animations and entities. Reusing these assets will save you a lot of time. Here is a partial list of things you can do with cinematics in Lumberyard, presented in no specific order:

- Use existing animations
- Loop animations
- Hide and unhide game entities
- Use *Mannequin* fragments
- Change transform properties
- Control audio
- Change environmental properties
- Make use of materials
- Evoke Full Screen mode
- Use cameras
- Implement lighting
- Add characters to scenes

Using cameras in cinematics

Cameras are often an important component in creating cut scenes. Cameras, like other Lumberyard objects, are accessible using the **RollupBar**. To add a camera, use the **RollupBar** | **Objects** | **Misc** | **Cameras** selection. There is, as you can see next, a host of properties you can change. These cameras can be hidden and used (change focus to) during cinematic playback.



Making your cinematics interactive

Earlier, we mentioned that Lumberyard cinematics, also known as cut scenes, can be interactive. This represents a powerful opportunity to engage your game's user in a new way. Lumberyard provides you with the ability to incorporate player interaction in the following ways.

- You can force the player to click a specific button. This tactic can be helpful if you want to invoke and loop a cut scene and wait for the user to signify they want to move on. This functionality can be created using the **Flow Graph Editor**.
- Push feedback to a user via their controller, such as with a rumble motor. To implement this, review **Game:ForceFeedback**-related nodes in the **Flow Graph Editor**.
- You can use the **Track View Editor** to allow players to rotate the active camera when a cut scene is being played.
- You can create **Flow Graphs** that give players the functionality to advance to or go back to a specific scene. This opens the door for some pretty creative gameplay.
- The last interactive component on our list is perhaps the most fun. Lumberyard calls it *Dead-Man Switch*. You can, using the **Flow Graph Editor**, create functionality to activate if the player fails to accomplish something during a cut scene, or fails to do something specific in a specified amount of time. The Lumberyard documentation (`http://docs.aws.amazon.com/lumberyard/latest/userguide/cinematics-interactivity-dead-man.html`) provides additional information on this functionality.

System streaming

System streaming is the process of controlling input and output data streams, that is, information to and from your game. The general concept is to stream data to memory when needed based on information requests. Animations, for example, are usually memory hogs. Having animations loaded into memory before they are needed is a sure way to quickly consume all of a system's available memory.

Although Lumberyard handles much of the streaming processes, game developers using Lumberyard to build their games should be familiar with this concept and the necessary interfaces, requirements, files, and applications associated with system streaming. Lumberyard's documentation provides some cursory information regarding streaming.

Memory handling

Amazon Lumberyard is intended for building AAA games, which implies high-quality, memory-intensive games. Memory, as it applies to our games, can be grouped into three broad categories: physical memory required for the game, memory requirements on the player's console or computer, and cloud-based storage requirements. In this section, we will briefly look at each of these three categories.

Physical memory for the game

It might not seem that how much physical disk space a game takes is an issue worth writing about. Remember, we are talking about AAA games, which are usually very large. If you are developing your game for a mobile device or a gaming console, then you will have to deal with additional restrictions. Mobile platform distribution networks, such as Apple's App Store and Google Play, have restrictions on the size of binary files uploaded to their respective stores. In additional, some consoles require a game to be playable within a specific amount of time once selected by the player.

There are techniques that you can use to address these concerns, such as progressive downloads where new levels, as an example, are loaded as needed, and not all at once.

Memory requirements for the targeted device

At some point, you will need to announce what the minimum system requirements are for your game. This will include the amount of disk space and RAM a player's computer must have. In addition, the computer's main and graphics chip sets should be described. Here is a comparison of blockbuster game titles from 1997, 2007, and 2017:

PC System Requirements for Popular Game Titles (titles suppressed)			
Spec	**1997**	**2007**	**2017**
Intel CPU	Pentium Processor	Pentium 4.2 GHz	Core i5-3470 3.2 GHz
RAM	16 MB	512 MB	8 GB
Graphics	SVGA with 1 MB VRAM	Nvidia GPU GeForce 6600	Nvidia GPU GeForce GTX 750 Ti
Hard Drive	80 MB	8 GB	80 GB
Direct X	N/A	DX 9	DX 12
Windows	Win 95	Win XP 32-bit	Win 10 64-bit

Depending on your game's requirements, Internet bandwidth might be worth announcing. You want your potential users to know if their computer system can handle the game. Imagine the frustration of purchasing a game just to find out that your system does not support the game. This concern has given rise to *test my system* and *check my PC* type links on game title sites. Here is an example for the game Overwatch: `http://www.systemrequirementslab.com/cyri/requirements/overwatch/12955`.

Another concern regarding system requirements is to make them high enough to support a high-quality gaming experience and low enough to attract a large user population. The challenge is to balance those two ends of the quality spectrum.

Cloud-based storage requirements

Game developers can take advantage of cloud-based storage solutions to support their games. In addition to leaderboards, networked games tend to use storage for lobbies, matches, and other multiplayer game functionality. We discussed cloud-based storage solutions in `Chapter 9`, *Employing Cloud Computing and Storage*.

Amazon Web Services

In `Chapter 7`, *Creating Multiplayer Game Play*, we explored the need for cloud-based services to support our Lumberyard games. We looked at AWS because of its integration with the Lumberyard game engine. You were guided through the process of creating your own AWS account for use with your Lumberyard game sessions. We also looked at GameLift, a core AWS gaming service. In `Chapter 9`, *Employing Cloud Computing and Storage*, we reviewed additional AWS solutions that are relevant to games we create with Lumberyard. Specifically, we looked at Cloud Canvas and **Simple Storage Service** (**S3**).

We will conclude this chapter with an overview of two additional cloud-based services from AWS that can represent a great way to support your game's functionality.

Simple Queue Service

When we develop games that use a message system, we usually have to write our own queuing system to manage the messages. Amazon offers a Simple Queue Service, or Amazon SQS for short, to help us manage the messages that our game uses. We can pass messages between game components, networked devices, game sessions, and more.

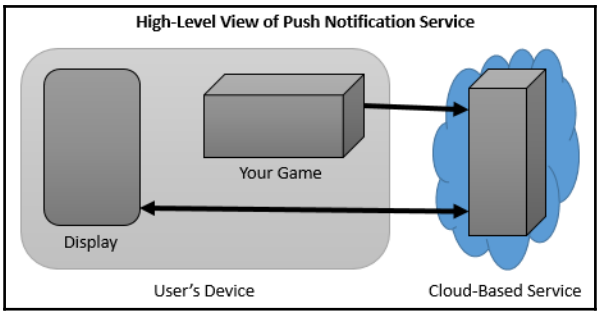
Amazon SQS uses text-formatted messages up to 256 KB in size. Using a queuing service can result in faster processing. The disaggregated queuing model has the production component pushing to the queue and other components, the ones intended to receive, processing incoming messages.

To learn more about Amazon's Simple Queue Service, you can visit the Amazon Simple Queue Service official site at `https://aws.amazon.com/sqs/`. You can also review the SQS Developer Guide at the following link: `http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/Welcome.html`.

Simple Notification Service

Amazon offers a Simple Notification Service, also referred to as *Amazon SNS*. If you are familiar with push notifications for mobile apps, then you already have some idea of what Amazon SNS can do. Push notification services require a cloud service to send notifications, receive push requests, and to manage the notifications. There is a complexity to push technology and, as the *Simple* suggests, Amazon's APIs make it relatively easy to implement this feature in our games.

The following figure illustrates how your Lumberyard game and a cloud-based push notification service such as Amazon SNS interact. Your game sends requests to the service and it generates and pushes the notification to the user's device based on how you set it up in your game.



If you can think of an instance where you want to provide notifications to players, then you can think of a use for Amazon SNS. Push notifications are frequently used for marketing, such as a *You haven't played in 1 week, come back and see what is new*-type messages. You can also use them to provide notifications that are related to guilds or matches.

There are several push notification services available; Amazon certainly was not the first to offer this service. What Amazon has done is integrate this service with other AWS such as the Simple Queue System.

To learn more about Amazon's Simple Notification Service, you can visit the Amazon Simple Notification Service official site at `https://aws.amazon.com/sns/`. You can also review the SNS Developer Guide at the following link:

```
http://docs.aws.amazon.com/sns/latest/api/Welcome.html
```

Summary

In this chapter, we went beyond the basics of just creating AAA games with Lumberyard. We looked at how Lumberyard can be used to create Virtual Reality applications. Our VR discussion included tips on how to test your VR game on an actual head-mounted device. We then looked at the Waf Build system, the publishing release system used by Lumberyard. Next, we explored how cut scenes can be constructed using Lumberyard's cinematics system. Later, the advanced topics of system streaming and memory handling were also covered. Finally, we ended the chapter with a review of two additional Amazon Web Services (Simple Query Service and Simple Notification Service).

You have come a long way since the first chapter. Lumberyard is an exciting new game engine and continues to mature at a rapid pace. You are now armed with enough knowledge to start building your own games with Lumberyard. There is so much more to learn and you'll continually learn as you develop. Have fun!

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X