CHAPTER 1

Introducing 2D Game Engine Development with JavaScript

Video games are complex, interactive, multimedia software systems. These systems must, in real time, process player input, simulate the interactions of semi-autonomous objects, and generate high-fidelity graphics and audio outputs, all while trying to engage the players. Attempts at building video games can quickly be overwhelmed by the need to be well versed in software development as well as in how to create appealing player experiences. The first challenge can be alleviated with a software library, or game engine, that contains a coherent collection of utilities and objects designed specifically for developing video games. The player engagement goal is typically achieved through careful gameplay design and fine-tuning throughout the video game development process. This book is about the design and development of a game engine; it will focus on implementing and hiding the mundane operations and supporting complex simulations. Through the projects in this book, you will build a practical game engine for developing video games that are accessible across the Internet.

A game engine relieves the game developers from simple routine tasks such as decoding specific key presses on the keyboard, designing complex algorithms for common operations such as mimicking shadows in a 2D world, and understanding nuances in implementations such as enforcing accuracy tolerance of a physics simulation. Commercial and well-established game engines such as Unity, Unreal Engine, and Panda3D present their systems through a graphical user interface (GUI). Not only does the friendly GUI simplify some of the tedious processes of game design such as creating and placing objects in a level, but more importantly, it ensures that these game engines are accessible to creative designers with diverse backgrounds who may find software development specifics distracting.

This book focuses on the core functionality of a game engine independent from a GUI. While a comprehensive GUI system can improve the end-user experience, the implementation requirements can also distract and complicate the fundamentals of a game engine. For example, issues concerning the enforcement of compatible data types in the user interface system, such as restricting objects from a specific class to be assigned as shadows receivers, are important to GUI design but are irrelevant to the core functionality of a game engine.

This book approaches game engine development from two important aspects: programmability and maintainability. As a software library, the interface of the game engine should facilitate programmability by game developers with well-abstracted utility methods and objects that hide simple routine tasks and support complex common operations. As a software system, the code base of the game engine should support maintainability with a well-designed infrastructure and well-organized source code systems that enable code reuse, ongoing system upkeep, improvement, and expansion.

This chapter describes the implementation technology and organization of the book. The discussion then leads you through the steps of downloading, installing, and setting up the development environment; guides you to build your first HTML5 application; and uses this first application development experience to explain the best approach to reading and learning from this book.

The Technologies

The goal of building a game engine that allows games to be accessible across the World Wide Web is enabled by freely available technologies.

JavaScript is supported by virtually all web browsers because an interpreter is installed on almost every personal computer in the world. As a programming language, JavaScript is dynamically typed, supports inheritance and functions as first-class objects, and is easy to learn with well-established user and developer communities. With the strategic choice of this technology, video games developed based on JavaScript can be accessible by anyone over the Internet through appropriate web browsers. Therefore, JavaScript is one of the best programming languages for developing video games for the masses.

While JavaScript serves as an excellent tool for implementing the game logic and algorithms, additional technologies in the form of software libraries, or application programming interfaces (APIs), are necessary to support the user input and media output requirements. With the goal of building games that are accessible across the Internet through web browsers, HTML5 and WebGL provide the ideal complementary input and output APIs.

HTML5 is designed to structure and present content across the Internet. It includes detailed processing models and the associated APIs to handle user input and multimedia outputs. These APIs are native to JavaScript and are perfect for implementing browser-based video games. While HTML5 offers a basic Scalable Vector Graphics (SVG) API, it does not support the sophistication demanded by video games for effects such as real-time lighting, explosions, or shadows. The Web Graphics Library (WebGL) is a JavaScript API designed specifically for the generation of 2D and 3D computer graphics through web browsers. With its support for OpenGL Shading Language (GLSL) and the ability to access the graphics processing unit (GPU) on client machines, WebGL has the capability of producing highly complex graphical effects in real time and is perfect as the graphics API for browser-based video games.

This book is about the concepts and development of a game engine where JavaScript, HTML5, and WebGL are simply tools for the implementation. The discussion in this book focuses on applying the technologies to realize the required implementations and does not try to cover the details of the technologies. For example, in the game engine, inheritance is implemented with the JavaScript object prototype chain; however, the merits of prototype-based scripting languages are not discussed. The engine audio cue and background music functionalities are based on the HTML5 `AudioContext` interface, and yet its range of capabilities is not described. The game engine objects are drawn based on WebGL texture maps, while the features of the WebGL texture subsystem are not presented. The specifics of the technologies would distract from the game engine discussion. The key learning outcomes of the book are the concepts and implementation strategies for a game engine and not the details of any of the technologies. In this way, after reading this book, you will be able to build a similar game engine based on any comparable set of technologies such as C# and MonoGame, Java and JOGL, C++ and Direct3D, and so on. If you want to learn more about or brush up on JavaScript, HTML5, or WebGL, please refer to the references in the "Technologies" section at the end of this chapter.

■ **Note** JavaScript supports inheritance via the language prototype chain mechanism. Technically, there is no class hierarchy to speak of. However, for clarity and simplicity, this book uses standard object-oriented terminology such as *superclass* and *subclass* to refer to parent-child inheritance relationships.

Setting Up Your Development Environment

The game engine you are going to build will be accessible through web browsers that could be running on any operating system (OS). The development environment you are about to set up is also OS agnostic. For simplicity, the following instructions are based on a Windows 7 or Windows 8 OS. You should be able to reproduce a similar environment with minor modifications in a Unix-based environment like the Apple OS X or Ubuntu.

Your development environment includes an integrated development environment (IDE) and a runtime web browser that is capable of hosting the running game engine. The most convenient systems we have found are the NetBeans IDE with the Google Chrome web browser as runtime environment. Here are the details:

- *IDE*: All projects in this book are based on the NetBeans IDE. You can download and install the bundle for HTML5 and PHP from `https://netbeans.org/downloads`.
- *Runtime environment*: You will execute your video game projects in the Google Chrome web browser. You can download and install this browser from `https://www.google.com/chrome/browser/`. Notice that Microsoft Internet Explorer 11 does not support HTML5 `AudioContext` and thus will not execute any projects after Chapter 4; in addition, Mozilla Firefox (version 39.0) does not support some of the GLSL shaders in Chapter 9.
- *Connector Google Chrome plug-in*: This is a Google Chrome extension that connects the web browser to the NetBeans IDE to support HTML5 development. You can download and install this extension from `https://chrome.google.com/webstore/detail/netbeans-connector/hafdlehgocfcodbgjnpecfajgkeejnaa`. The download will automatically install the plug-in to Google Chrome. You may have to restart your computer to complete the installation.
- *glMatrix math library*: This is a library that implements the foundational mathematic operations. You can download this library from `http://glMatrix.net`. You will integrate this library into your game engine in Chapter 3, so more details will be provided there.

Notice that there are no specific system requirements to support the JavaScript programming language, HTML5, or WebGL. All these technologies are embedded in the web browser runtime environment.

■ **Note** As mentioned, we chose NetBeans-based development environment because we found it to be the most convenient. There are many other alternatives that are also free, including and not limited to IntelliJ IDEA, Eclipse, and Sublime.

Downloading and Installing JavaScript Syntax Checker

We have found JSLint to be an effective tool in detecting potential JavaScript source code errors. You can download and install JSLint as a plug-in to the NetBeans IDE with the following steps:

- Download it from `http://plugins.netbeans.org/plugin/40893/jslint`. Make sure to take note of the location of the downloaded file.
- Start NetBeans, select Tools ➤ Plugins, and go to the Downloaded tab.
- Click the Add Plugins button and search for the downloaded file from step 1. Double-click this file to install the plug-in.

The following are some useful references for working with JSLint:

- For instructions on how to work with JSLint, see `http://www.jslint.com/help.html`.
- For details on how JSLint works, see `http://plugins.netbeans.org/plugin/40893/jslint`.

Working in the NetBeans Development Environment

The NetBeans IDE is easy to work with, and the projects in this book require only the editor and debugger. To open a project, select File ➤ Open Projects. Once a project is open, you need to become familiarize with three basic windows, as illustrated in Figure 1-1.

- *Projects window*: This window displays the source code files of the project.
- *Editor window*: This window displays and allows you to edit the source code of your project. You can select the source code file to work with by double-clicking the corresponding file name in the Projects window.
- *Action Items window*: This window displays the error message output from the JSLint checker.

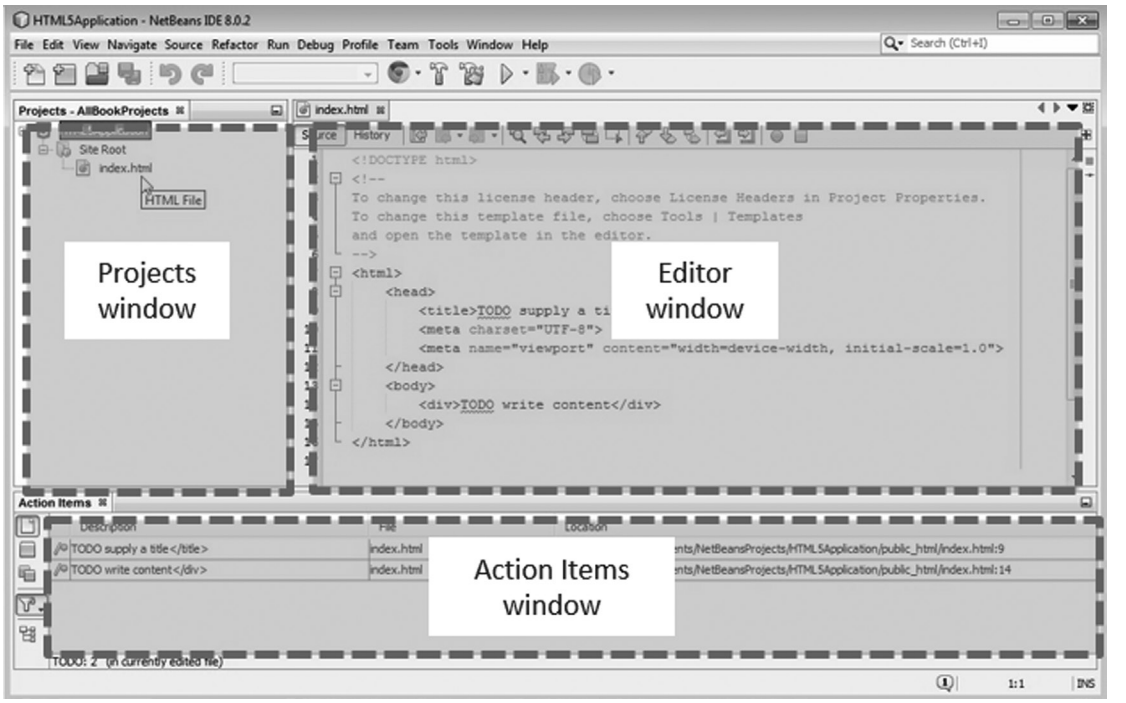

Figure 1-1. *The NetBeans IDE*

■ **Note** If you cannot see a window in the IDE, you can click the Window menu and select the name of the missing window to cause it to appear. For example, if you cannot see the Projects window in the IDE, you can select Window ➤ Projects to open it.

Creating an HTML5 Project in NetBeans

You are now ready to create your first HTML5 project.

- Start NetBeans. Select File ➤ New Project (or press Ctrl+Shift+N), as shown in Figure 1-2. A New Project window will appear.

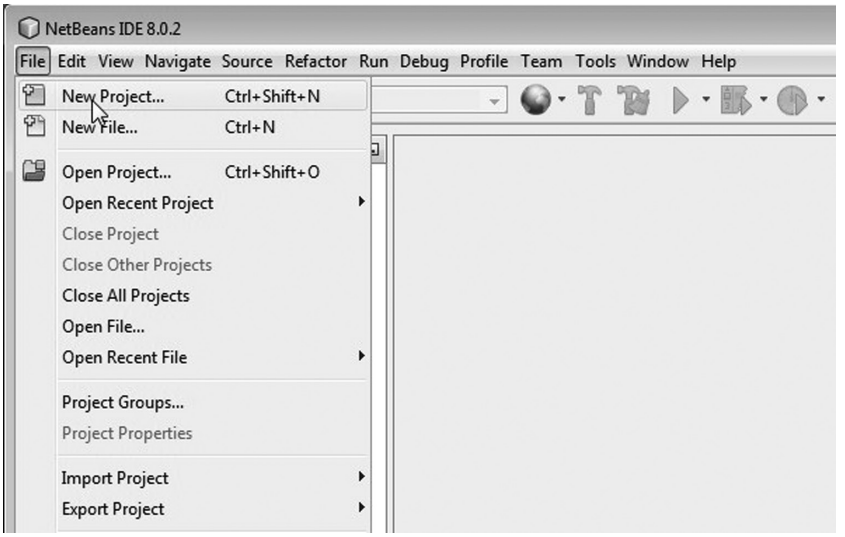

Figure 1-2. *Creating a new project*

- In the New Project window, select HTML5 in the Categories section, and select HTML5 Application from the Projects section, as shown in Figure 1-3. Click the Next button to bring up the project configuration window.

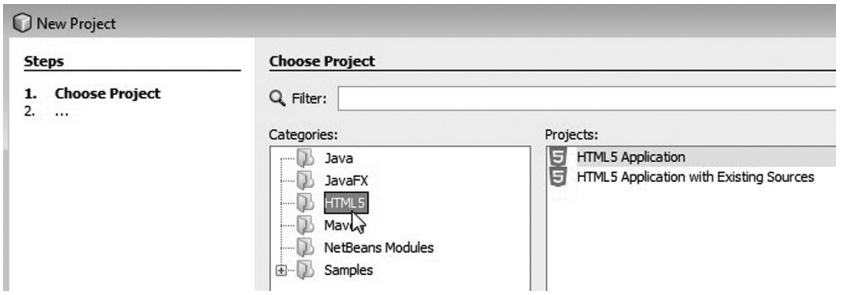

Figure 1-3. *Selecting the HTML5 project*

- As shown in Figure 1-4, enter the name and location of the project, and click the Finish button to create your first HTML5 project.

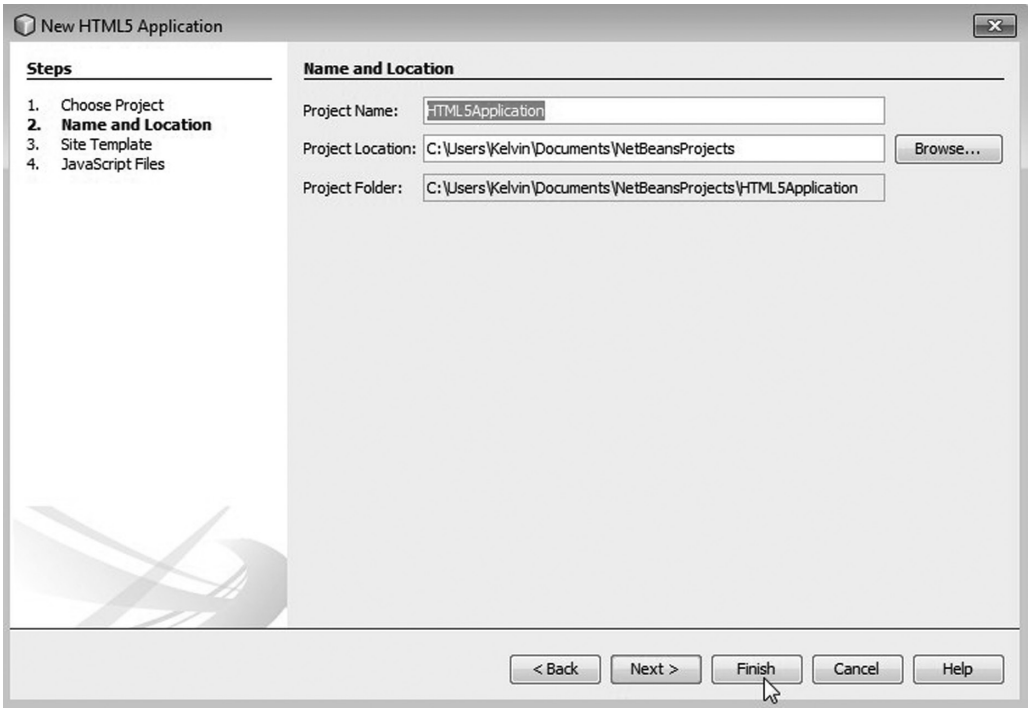

Figure 1-4. *Naming the project*

NetBeans will generate the template of a simple and complete HTML5 application project for you. Your IDE should look similar to Figure 1-5.

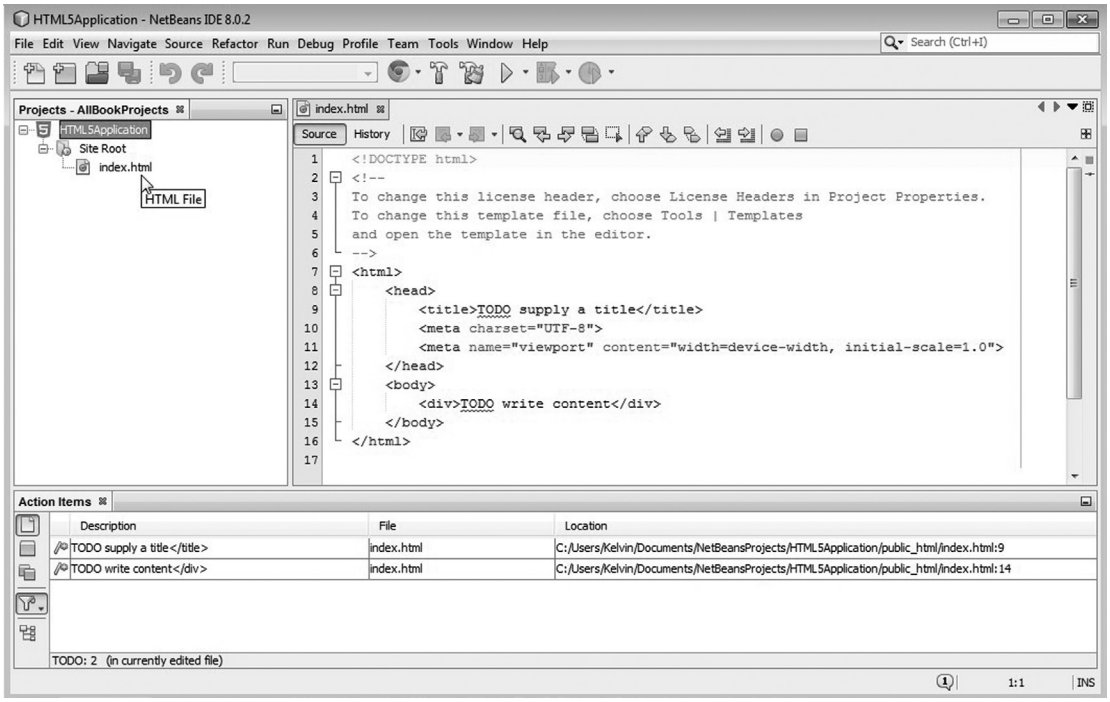

Figure 1-5. *The HTML5 application project*

By selecting and double-clicking the `index.html` file in the Projects window, you can open it in the Editor window and observe the content of the file. The contents are as follows:

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

The first line declares the file to be an HTML file. The block that follows within the `<!--` and `-->` tags is a comment block. The complementary `<html></html>` tags contain all the HTML code. In this case, the template defines the `head` and `body` sections. The `head` sets the title of the web page, and the `body` is where all the content for the web page will be located.

You can run this project by selecting Run ➤ Run Project or by pressing the F6 key. Figure 1-6 shows an example of what the default project looks like when you run it.

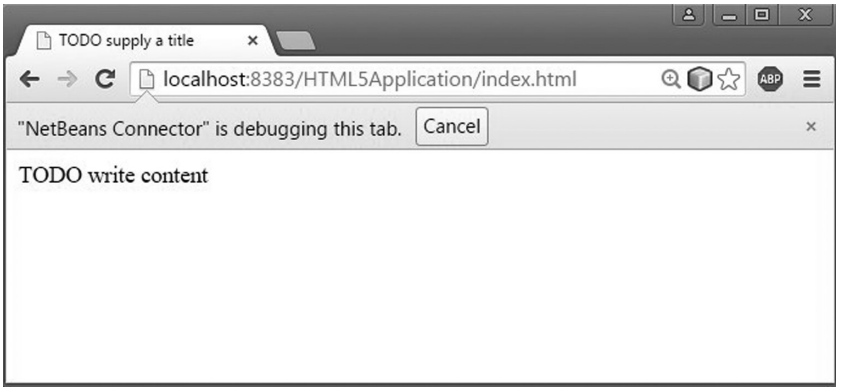

Figure 1-6. *Running the simple HTML5 project*

■ **Note** As will be explained in the next chapter, you cannot double-click the `index.html` file to run the project. You must either select Run ➤ Run Project, press on the F6 key, or click the green triangle button.

To stop the program, either close the web page or click the Cancel button in the browser to stop NetBeans from tracking the web page. You have successfully run your first HTML5 project. You can use this project to understand the IDE environment.

The Relationship Between the Project Files and the File System

Navigate to the `HTML5Application` project location on your file system, for example with the Explorer OS utility in Windows 7. You can observe that in the project folder NetBeans has generated the `nbProject`, `public_html`, and `test` folders. Table 1-1 summarizes the purpose of these folders and the `index.html` file.

Table 1-1. *Folders and Files in a NetBeans HTML5 Project*

NetBeans HTML5 project: folder/file	Purpose
`nbProject/`	This folder contains the IDE configuration files. You will not modify any of the files in this folder.
`public_html/`	This is the root folder of your project. Source code and assets from your project will be created in this folder.
`public_html/index.html`	This is the default entry point for your web site. This file will be modified to load JavaScript source code files.
`test/`	This is the default folder for unit testing source code files. This folder is not used in this book and has been removed from all the projects.

How to Use This Book

This book guides you through the development of a game engine by building projects similar to the one you have just experienced. Each chapter covers an essential component of a typical game engine, and the sections in each chapter describe the important concepts and implementation projects that construct the corresponding component. Throughout the text, the project from each section builds upon the results from the projects that precede it. While this makes it a little challenging to skip around in the book, it will give you practical experience and a solid understanding of how the different concepts relate. In addition, rather than always working with new and minimalistic projects, you gain experience with building larger and more interesting projects while integrating new functionality into your expanding game engine.

The projects start with demonstrating simple concepts, such as drawing a simple square, but evolve quickly into presenting more complex concepts, such as working with user-defined coordinate systems and implementing pixel-accurate collision detection. Initially, as you have experienced in building the first HTML5 application, you will be guided with detailed steps and complete source code listings. As you become familiar with the development environment and the technologies, the guides and source code listings accompanying each project will shift to highlight on the important implementation details. Eventually, as the complexity of the projects increases, the discussion will focus only on the vital and relevant issues, while straightforward source code changes will not be mentioned.

The final code base, which you will have developed incrementally over the course of the book, is a complete and practical game engine; it's a great platform on which you can begin building your own 2D games. This is exactly what the last chapter of the book does, leading you from the conceptualization to design to implementation of a casual 2D game.

There are several ways for you to follow along with this book. The most obvious is to enter the code into your project as you follow each step in the book. From a learning perspective, this is the most effective way to absorb the information presented; however, we understand that it may not be the most realistic because of the amount of code or debugging this approach may require. Alternatively, we recommend that you run and examine the source code of the completed project when you begin a new section. Doing so lets you preview the current section's project, gives you a clear idea of the end goal, and lets you see what the project is trying to achieve. You may also find the completed project code useful when you have problems while building the code yourself, because you can compare your code with the completed project's code during difficult debugging situations.

■ **Note** We have found the WinMerge program (`http://winmerge.org/`) to be an excellent tool for comparing source code files and folders. Mac users can check out the FileMerge utility for a similar purpose.

Finally, after completing a project, we recommend that you compare the behavior of your implementation with the completed implementation provided. By doing so, you can observe whether your code is behaving as expected.

How Do You Make a Great Video Game?

While the focus of this book is on the design and implementation of a game engine, it is important to appreciate how different components can contribute to the creation of a fun and engaging video game. Beginning in Chapter 4, a "Game Design Consideration" section is included at the end of each chapter to relate the functionality of the engine component to elements of game designs. This section presents the framework for these discussions.

It's a complex question, and there's no exact formula for making a video game that people will love to play, just as there's no exact formula for making a movie that people will love to watch. We've all seen big-budget movies that look great and feature top acting, writing, and directing talent but that bomb at the box office, and we've all seen big-budget games from major studios that fail to capture the imaginations of players. By the same token, movies by unknown directors can grab the world's attention, and games from small, unknown studios can take the market by storm.

While no explicit instructions exist for making a great game, a number of elements work together in harmony to create a final experience greater than the sum of its parts, and all game designers must successfully address each of them in order to produce something worth playing. The elements include the following:

- *Technical design*: This includes all game code and the game platform and is generally not directly exposed to players; rather, it forms the foundation and scaffolding for all aspects of the game experience. This book is primarily focused on issues related to the technical design of games, including specific tasks such as the lines of code required to draw elements on the screen and more architectural considerations such as determining the strategy for how and when to load assets into memory. Technical design issues impact the player experience in many ways (for example, the number of times a player experiences "loading" delays during play or how many frames per second the game displays), but the technical design is typically invisible to players because it runs under what's referred to as the presentation layer, or all of the audiovisual and/or haptic feedback the player encounters during play.

- *Game mechanic(s)*: The game mechanic is an abstract description of what can be referred to as the foundation of play for a given game experience. Types of game mechanics include puzzles, dexterity challenges such as jumping or aiming, timed events, combat encounters, and the like. The game mechanic is a framework; specific puzzles, encounters, and game interactions are implementations of the framework. A real-time strategy (RTS) game might include a resource-gathering mechanic, for example, where the mechanic might be described as "Players are required to gather specific types of resources and combine them to build units which they can use in combat." The specific implementation of that mechanic (how players locate and extract the resources in the game, how they transport them from one place to another, and the rules for combining resources to produce units) is an aspect of system design, level design, and the interaction model (described later in this section).

- *Systems design*: The internal rules and logical relationships that provide structured challenge to the core game mechanic are referred to as the game's systems design. Using the previous RTS example, a game might require players to gather a certain amount of metal ore and combine it with a certain amount of wood to make a game object; the specific rules for how many of each resource is required to make the objects and the unique process for creating the objects (for example, objects can be produced only in certain structures on the player's base and take *x* number of minutes to appear after the player starts the process) are aspects of systems design. Casual games may have basic systems designs. The unexpected global phenomenon Flappy Bird from .GEARS Studio, for example, is a game with few systems and low complexity, while major genres like RTS games may have deeply complex and interrelated systems designs created and balanced by entire teams of designers. Game systems designs are often where the most hidden complexity of game design exists; as designers go through the exercise of defining all variables that contribute to an implementation of a game

mechanic, it's easy to become lost in a sea of complexity and balance dependencies. Systems that appear fairly simple to players may require many components working together and balanced perfectly against each other, and underestimating systems complexity is perhaps one of the biggest pitfalls encountered by new (and veteran!) game designers. Until you know what you're getting into, always assume the systems you create will prove to be considerably more complex than you anticipate.

- *Level design*: A game's level design reflects the specific ways each of the other eight elements combine within the context of individual "chunks" of gameplay, where players must complete a certain chunk of objectives before continuing to the next section (some games may have only one level, while others will have dozens). Level designs within a single game can all be variations of the same core mechanic and systems design (games like Tetris and Bejeweled are examples of games with many levels all focusing on the same mechanic), while other games will mix and match mechanics and systems designs for variety among levels. Most games feature one primary mechanic and a game-spanning approach to systems design and will add minor variations between levels to keep things feeling fresh (changing environments, changing difficulty, adding time limits, increasing complexity, and the like), although occasionally games will introduce new levels that rely on completely separate mechanics and systems to surprise players and hold their interest. Great level design in games is a balance between creating "chunks" of play that showcase the mechanic and systems design and changing enough between these chunks to keep things interesting for players as they progress through the game (but not changing so much between chunks that the gameplay feels disjointed and disconnected).

- *Interaction model*: The interaction model is the combination of keys, buttons, controller sticks, touch gestures, and so on, used to interact with the game to accomplish tasks and the graphical user interfaces that support those interactions within the game world. Some game theorists break the game's user interface (UI) design into a separate category (game UI includes things such as menu designs, item inventories, heads-up displays [HUDs]), but the interaction model is deeply connected to UI design, and it's good practice to think of these two elements as inseparable. In the case of the RTS game referenced earlier, the interaction model includes the actions required to select objects in the game, to move those objects, to open menus and manage inventories, to save progress, to initiate combat, and to queue build tasks. The interaction model is completely independent of the mechanic and systems design and is concerned only with the physical actions the player must take to initiate behaviors (for example, click mouse button, press key, move stick, scroll wheel); the UI is the audiovisual or haptic feedback connected to those actions (onscreen buttons, menus, statuses, audio cues, vibrations, and the like).

- *Game setting*: Are you on an alien planet? In a fantasy world? In an abstract environment? The game setting is a critical part of the game experience and, in partnership with the audiovisual design, turns what would otherwise be a disconnected set of basic interactions into an engaging experience with context. Games settings need not be elaborate to be effective; the game Candy Crush from King has a rather simple setting with a thin narrative wrapper, but the combination of setting, audiovisual design, and level design are uniquely well-matched and contribute significantly to the millions of hours players invest in the experience each month (as of 2015).

- *Visual design*: Video games exist in a largely visual medium, so it's not surprising that companies frequently spend as much or more on the visual design of their games as they spend on the technical execution of the code. Large games are aggregations of thousands of visual assets, including environments, characters, objects, animations, and cinematics; even small casual games generally ship with hundreds or thousands of individual visual elements. Each object a player interacts with in the game must be a unique asset, and if that asset includes more complex animation than just moving it from one location on the screen to another or changing the scale or opacity, the object most likely will need to be animated by an artist. Game graphics need not be photorealistic or stylistically elaborate to be visually excellent or to effectively represent the setting (many games intentionally utilize a simplistic visual style), but the best games consider art direction and visual style to be core to the player experience, and visual choices will be intentional and well-matched to the game setting and mechanic.
- *Audio design*: This includes music and sound effects, ambient background sounds, and all sounds connected to player actions (select/use/swap item, open inventory, invoke menu, and the like). Audio design functions hand-in-hand with visual design to convey and reinforce game setting, and many new designers significantly underestimate the impact of sound to immerse players into game worlds. Imagine *Star Wars* (for example) without the music, the light saber sound effect, Darth Vader's breathing, or R2D2's characteristic beeps; the audio effects and musical score are as fundamental to the experience as the visuals.
- *Meta-game*: The meta-game centers on how individual objectives come together to propel players through the game experience (often via scoring, unlocking individual levels in sequence, playing through a narrative, and the like). In many modern games, the meta-game is the narrative arc or story; players often don't receive a "score" per se but rather reveal a linear or semi-linear story as they progress through game levels, driving forward to complete the story. Other games (especially social and competitive games) involve players "leveling up" their characters, which can happen as a result of playing through a game-spanning narrative experience or by simply venturing into the game world and undertaking individual challenges that grant experience points to characters. Other games, of course, continue focusing on scoring points or winning rounds against other players.

The magic of video games typically arises from the interplay between these nine elements, and the most successful games finely balance each as part of a unified vision to ensure a harmonious experience; this balance will always be unique to each individual effort and is found in games ranging from King's Candy Crush to Bioware's Mass Effect. The core game mechanic in many successful games is often a variation on one or more fairly simple, common themes (Angry Birds, for example, is a relatively basic projectile launching game), but the visual design, narrative context, audio effects, interactions, and progression system work together with the game mechanic to create a unique experience that's considerably more engaging than the sum of its individual parts, making players want to return to it again and again. Great games range from the simple to the complex, but they all feature an elegant balance of supporting design elements.

References

The examples in this book are created with the assumptions that you understand data encapsulation, inheritance, and basic data structures, such as linked lists and dictionaries, and are comfortable working with the fundamentals of algebra and geometry, particularly linear equations and coordinate systems. Many examples in this book apply and implement concepts in computer graphics and linear algebra. These concepts warrant much more in-depth examinations. Interested readers can learn more about these topics in other books.

- Computer graphics:
  - Shirley, Ashikhmin, and Marschner. *Fundamentals of Computer Graphics*, 3rd edition. A. K. Peters, 2009.
  - Angle and Shreiner. *Interactive Computer Graphics: A Top Down Approach with WebGL*. Pearson Education, 2014.
- Linear algebra:
  - Johnson, Riess, and Arnold. *Introduction to Linear Algebra*, 5th edition. Addison-Wesley, 2002.
  - Anton and Rorres. *Elementary Linear Algebra: Applications Version, 11th edition*. Wiley, 2013.

Technologies

The following list offers links for obtaining additional information on technologies used in this book:

- *JavaScript*: `http://www.w3schools.com/js`
- *HTML5*: `http://www.w3schools.com/html/html5_intro.asp`
- *WebGL*: `https://www.khronos.org/webgl`
- *OpenGL*: `https://www.opengl.org`
- *NetBeans*: `https://netbeans.org`
- *Chrome*: `https://www.google.com/chrome`
- *glMatrix*: `http://glMatrix.net`
- *JSLint*: `http://www.jslint.com`