Build Your Own 2D Game Engine and Create Great Web Games

Using HTML5, JavaScript, and WebGL2

Second Edition

Kelvin Sung
Jebediah Pavleas
Matthew Munson
Jason Pace

Apress®

Build Your Own 2D Game Engine and Create Great Web Games

Using HTML5, JavaScript, and WebGL2

Second Edition

Kelvin Sung
Jebediah Pavleas
Matthew Munson
Jason Pace

With
Original Dye character designs by Nathan Evers
Other game character and art design by Kasey Quevedo
Figures and illustrations by Clover Wai

Apress®

Build Your Own 2D Game Engine and Create Great Web Games: Using HTML5, JavaScript, and WebGL2

Kelvin Sung
Bothell, WA, USA

Jebediah Pavleas
Kenmore, WA, USA

Matthew Munson
Lake Forest Park, WA, USA

Jason Pace
Portland, OR, USA

ISBN-13 (pbk): 978-1-4842-7376-0
ISBN-13 (electronic): 978-1-4842-7377-7
https://doi.org/10.1007/978-1-4842-7377-7

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Spandana Chatterjee
Development Editor: Laura Berendson
Coordinating Editors: Shrikant Vishwakarma

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484273760. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To my wife, Clover, and our girls, Jean and Ruth, for completing my life.
—Kelvin Sung

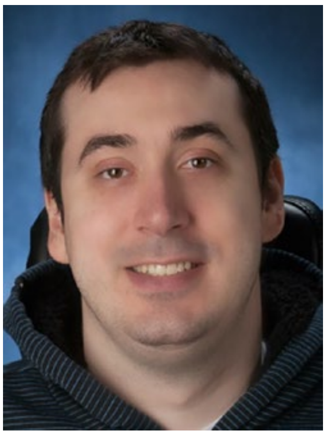
To my family, for their eternal support throughout my life.
—Jebediah Pavleas

To my mom, Linda, for showing me the value of having fun at work.
—Jason Pace

Table of Contents

About the Authors

Kelvin Sung is a Professor with the Computing and Software Systems Division at the University of Washington Bothell (UWB). He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. Kelvin's background is in computer graphics, hardware, and machine architecture. He came to UWB from Alias|Wavefront (now part of Autodesk), where he played a key role in designing and implementing the Maya Renderer, an Academy Award–winning image generation system. At UWB, funded by Microsoft Research and the National Science Foundation, Kelvin's work focuses on the intersection of video game mechanics, solutions to real-world problems, and supports for remote collaboration. Together with his students and colleagues, Kelvin has co-authored five books: one on computer graphics (*Essentials of Interactive Computer Graphics: Concepts and Implementation*, A.K. Peters, 2008) and the others on 2D game engines (*Learn 2D Game Development with C#*, Apress, 2013; *Build Your Own 2D Game Engine and Create Great Web Games*, Apress, October 2015; *Building a 2D Game Physics Engine*, Apress, 2016; and *Basic Math for Game Development with Unity 3D*, Apress, 2019).



Jebediah Pavleas received his Master of Science Degree in Computer Science and Software Engineering from the University of Washington Bothell (UWB) in 2016. He also received a Bachelor of Science Degree from UWB in 2012 and was the recipient of the Chancellor's Medal for his class. During his graduate program, Jebediah interned for Microsoft Research's Enable team where he contributed to their Eye-Gaze Wheelchair project (a wheelchair driven with only your eyes for those with ALS). He has co-authored three books on 2D games and game engines (*Learn 2D Game Development with C#*, Apress, 2013; *Build Your Own 2D Game Engine and Create Great Web Games*, Apress, October 2015; *Building a 2D Game Physics Engine*, Apress, 2016). During his time at UWB, his projects included an interactive math application that utilizes Microsoft's Kinect sensor to teach algebra called Kinect Math. Relating to this and other projects, he co-authored publications in *IEEE Computer* and *The Journal of Computing Sciences in Colleges* (CCSC). Jebediah enjoys designing, building, and playing games of all kinds as well as adapting technology for improved accessibility for himself and others.

Matthew Munson is a graduate student in the Computer Science and Software Engineering program at the University of Washington Bothell. He received undergraduate degrees in Computer Science and Software Engineering and Mechanical Engineering at the University of Washington Bothell in 2020. Matthew is interested in operating system development, networking, and embedded systems. As a research assistant, Matthew used cloud computing to analyze years of audio data recorded by hydrophones off the Oregon coast. This data was used to study the effects of climate change and shipping noise on marine mammals. Currently, Matthew is working on a networked augmented reality library that focuses on allowing users to view the same virtual scene from different perspectives.

Jason Pace contributed to a wide range of games as a producer, designer, and creative director over 15 years in the interactive entertainment industry, from ultra-casual puzzlers on mobile to Halo on Xbox. As a designer, Jason builds game mechanics and systems that start from a simple palette of thoughtful interactions (known as the core gameplay loop), progressively introducing variety and complexity to create interactive experiences that engage and delight players while maintaining focus on what makes each e-game uniquely fun.

About the Technical Reviewers

Yusuf Pisan is an Associate Teaching Professor in the School of Computing & Software Systems Division at the University of Washington Bothell. Previously, he has worked at the University of Technology, Sydney, and has been a visiting professor at Harvey Mudd College, University of Southern California, Worcester Polytechnic Institute (WPI), and IT University of Copenhagen (ITU).

His research interests include enabling technologies for computer games, the design of virtual environments that support collaborative work, and computer science education. He founded the Australasian Conference on Interactive Entertainment conference series and helped foster the Australian games community. His list of publications can be found at Google Scholar.

Yusuf has a Ph.D. in Artificial Intelligence from Northwestern University. Before moving to Seattle in 2017, Yusuf lived in the Chicago area for 10 years and Sydney for 20 years.

For more information, see `https://pisanorg.github.io/yusuf/`.

Yogendra Sharma is a developer with experience in the architecture, design, and development of scalable and distributed applications, with a core interest in Microservices and Spring. He currently works as an IoT and Cloud Architect at Intelizign Engineering Services Pvt. Ltd., Pune.

He also has hands-on experience in technologies such as AWS, IoT, Python, J2SE, J2EE, NodeJS, VueJs, Angular, MongoDB, and Docker.

He constantly explores technical novelties, and he is open-minded and eager to learn about new technologies and frameworks. He has reviewed several books and video courses published by Packt.

Acknowledgments

This book project was a direct result of the authors learning from building games for the Game-Themed CS1/2: Empowering the Faculty project, funded by the Transforming Undergraduate Education in Science, Technology, Engineering, and Mathematics (TUES) Program, National Science Foundation (NSF) (award number DUE-1140410). We would like to thank NSF officers Suzanne Westbrook for believing in our project and Jane Prey, Valerie Bar, and Paul Tymann for their encouragements.

This second edition is encouraged by many students and collaborators. In particular, students from CSS452: Game Engine Development (see `https://myuwbclasses.github.io/CSS452/`) at the University of Washington Bothell have been the most critical, demanding, and yet supportive. Through the many games and API extension projects (see `https://html5gameenginegroup.github.io/GTCS-Engine-Student-Projects/`), it became clear that updates are required of the JavaScript and WebGL (Web Graphics Library) versions, the bottom-line synchronization mechanism, and, most significantly, the coverage of the physics engine. Fernando Arnez, our co-author from the first edition, taught us JavaScript. Yaniv Schwartz pointed us toward JavaScript async/await and promise. The discussions and collaborations with Huaming Chen and Michael Tanaya contributed directly to the chapter on game engine physics. Akilas Mebrahtom and Donald Hawkins constructed the extra example at the end of Chapter 9 illustrating potential presets for commonly encountered physical materials. The audio volume control was first investigated and integrated by Kyla NeSmith. Nicholas Carpenetti and Kyla NeSmith developed a user interface API for the initial game engine, which unfortunately did not make it into this edition. These and countless other feedbacks have contributed to the quality and improvements of the book's content.

The hero character Dye and many of the visual and audio assets used throughout the example projects of the book are based on the Dye Hard game, designed for teaching concepts of objects and object-oriented hierarchy. The original Dye Hard development team members included Matthew Kipps, Rodelle Ladia, Chuan Wang, Brian Hecox, Charles Chiou, John Louie, Emmett Scout, Daniel Ly, Elliott White, Christina Jugovic, Rachel Harris, Nathan Evers, Kasey Quevedo, Kaylin Norman-Slack, David Madden, Kyle Kraus, Suzi Zuber, Aina Braxton, Kelvin Sung, Jason Pace, and Rob Nash. Kyle Kraus composed the background music used in the Audio Support project from Chapter 4, originally for the Linx game, which was designed to teach loops. The background audio for the game in Chapter 12 was composed by David Madden and arranged by Aina Braxton. Thanks to Clover Wai for the figures and illustrations.

We also want to thank Spandana Chatterjee for believing in our ideas, her patience, and continual efficient and effective support. A heartfelt thank-you to Mark Powers, for his diligence and lightning-fast email responses. Mark should learn about and consider the option of sleeping some of the time. Nirmal Selvaraj organized everything and ensured proper progress was ongoing.

Finally, we would like to thank Yusuf Pisan for his insightful, effective, and, above all, quick turnaround for the technical review.

All opinions, findings, conclusions, and recommendations in this work are those of the authors and do not necessarily reflect the views of the sponsors.

Introduction

Welcome to *Build Your Own 2D Game Engine and Create Great Web Games*. Because you have picked up this book, you are likely interested in the details of a game engine and the creation of your own games to be played over the Internet. This book teaches you how to build a 2D game engine by covering the involved technical concepts, demonstrating sample implementations, and showing you how to organize the large number of source code and asset files to support game development. This book also discusses how each covered technical topic area relates to elements of game design so that you can build, play, analyze, and learn about the development of 2D game engines and games. The sample implementations in this book are based on HTML5, JavaScript, and WebGL2, which are technologies that are freely available and supported by virtually all web browsers. After reading this book, the game engine you develop and the associated games will be playable through a web browser from anywhere on the Internet.

This book presents relevant concepts from software engineering, computer graphics, mathematics, physics, game development, and game design—all in the context of building a 2D game engine. The presentations are tightly integrated with the analysis and development of source code; you'll spend much of the book building game-like concept projects that demonstrate the functionality of game engine components. By building on source code introduced early on, the book leads you on a journey through which you will master the basic concepts behind a 2D game engine while simultaneously gaining hands-on experience developing simple but working 2D games. Beginning from Chapter 4, a "Design Considerations" section is included at the end of each chapter to relate the covered technical concepts to elements of game design. By the end of the book, you will be familiar with the concepts and technical details of 2D game engines, feel competent in implementing functionality in a 2D game engine to support commonly encountered 2D game requirements, and capable of considering game engine technical topics in the context of game design elements in building fun and engaging games.

New in the Second Edition

The key additions to the second edition include JavaScript language and WebGL API update and dedicated chapters with substantial details on physics and particle systems components.

All examples throughout the entire book are refined for the latest features of the JavaScript language. While some updates are mundane, for example, prototype chain syntax replacements, the latest syntax allows significant improvements in overall presentation and code readability. The new and much cleaner asynchronous support facilitated a completely new resource loading architecture with a single synchronization point for the entire engine (Chapter 4). The WebGL context is updated to connect to WebGL 2.0. The dedicated chapters allow more elaborate and gradual introduction to the complex physics and particle systems components. Detailed mathematical derivations are included where appropriate.

Who Should Read This Book

This book is targeted toward programmers who are familiar with basic object-oriented programming concepts and have a basic to intermediate knowledge of an object-oriented programming language such as Java or C#. For example, if you are a student who has taken a few introductory programming courses, an experienced developer who is new to games and graphics programming, or a self-taught programming enthusiast, you will be able to follow the concepts and code presented in this book with little trouble. If you're new to programming in general, it is suggested that you first become comfortable with the JavaScript programming language and concepts in object-oriented programming before tackling the content provided in this book.

Assumptions

You should be experienced with programming in an object-oriented programming language, such as Java or C#. Knowledge and expertise in JavaScript would be a plus but are not necessary. The examples in this book were created with the assumption that you understand data encapsulation and inheritance. In addition, you should be familiar with basic data structures such as linked lists and dictionaries and be comfortable working with the fundamentals of algebra and geometry, particularly linear equations and coordinate systems.

Who Should Not Read This Book

This book is not designed to teach readers how to program, nor does it attempt to explain the intricate details of HTML5, JavaScript, or WebGL2. If you have no prior experience developing software with an object-oriented programming language, you will probably find the examples in this book difficult to follow.

On the other hand, if you have an extensive background in game engine development based on other platforms, the content in this book will be too basic; this is a book intended for developers without 2D game engine development experience. However, you might still pick up a few useful tips about 2D game engine and 2D game development for the platforms covered in this book.

Organization of This Book

This book teaches how to develop a game engine by describing the foundational infrastructure, graphics system, game object behaviors, camera manipulations, and a sample game creation based on the engine.

Chapters 2–4 construct the foundational infrastructure of the game engine. Chapter 2 establishes the initial infrastructure by separating the source code system into folders and files that contain the following: JavaScript-specific core engine logics, WebGL2 GLSL–specific shader programs, and HTML5-specific web page contents. This organization allows ongoing engine functionality expansion while maintaining localized source code system changes. For example, only JavaScript source code files need to be modified when introducing enhancements to game object behaviors. Chapter 3 builds the drawing framework to encapsulate and hide the WebGL2 drawing specifics from the rest of the engine. This drawing framework allows the development of game object behaviors without being distracted by how they are drawn. Chapter 4 introduces and integrates core game engine functional components including game loop, keyboard input, efficient resource and game-level loading, and audio support.

Chapters 5–7 present the basic functionality of a game engine: drawing system, behavior and interactions, and camera manipulation. Chapter 5 focuses on working with texture mapping, including sprite sheets, animation with sprite sheets, and the drawing of bitmap fonts. Chapter 6 puts forward abstractions for game objects and their behaviors including per-pixel-accurate collision detection. Chapter 7 details the manipulation and interactions with the camera including programming with multiple cameras and supporting mouse input.

Chapters 8–11 elevate the introduced functionality to more advanced levels. Chapter 8 covers the simulation of 3D illumination effects in 2D game scenes. Chapter 9 discusses physically based behavior simulations. Chapter 10 presents the basics of particle systems that are suitable for modeling explosions. Chapter 11 examines more advanced camera functionality including infinite scrolling through tiling and parallax.

Chapter 12 summarizes the book by leading you through the design of a complete game based on the game engine you have developed.

Code Samples

Every chapter in this book includes examples that let you interactively experiment with and learn the new materials. You can access the source code for all the projects, including the associated assets (images, audio clips, or fonts), by clicking the **Download Source Code** button located at `www.apress.com/9781484273760`. You should see a folder structure that is organized by chapter numbers. Within each folder are subfolders containing Visual Studio Code (VS Code) projects that correspond to sections of this book.

CHAPTER 1

Introducing 2D Game Engine Development with JavaScript

Video games are complex, interactive, multimedia software systems. They must, in real time, process player input, simulate the interactions of semiautonomous objects, and generate high-fidelity graphics and audio outputs, all while trying to keep players engaged. Attempts at building a video game can quickly become overwhelming with 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 of the engine while supporting many 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 game developers from having to implement 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

© Kelvin Sung, Jebediah Pavleas, Matthew Munson, and Jason Pace 2022
K. Sung et al., *Build Your Own 2D Game Engine and Create Great Web Games*,
https://doi.org/10.1007/978-1-4842-7377-7_1

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 shadow 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 yet 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 this book. The discussion 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 class functionality which is based on 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.

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 10 OS. You should be able to reproduce a similar environment with minor modifications in a Unix-based environment like MacOS 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 is the Visual Studio Code (VS Code) IDE with the Google Chrome web browser as runtime environment. Here are the details:

- **IDE**: All projects in this book are based on VS Code IDE. You can download and install the program from `https://code.visualstudio.com/`.
- **Runtime environment**: You will execute your video game projects in the Google Chrome web browser. You can download and install this browser from `www.google.com/chrome/browser/`.
- **glMatrix math library**: This is a library that implements the foundational mathematical 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 the VS Code–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 NetBeans, IntelliJ IDEA, Eclipse, and Sublime.

Downloading and Installing JavaScript Syntax Checker

We have found ESLint to be an effective tool in detecting potential JavaScript source code errors. You can integrate ESLint into VS Code with the following steps:

- Go to `https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint` and click install.
- You will be prompted to open VS Code and may need to click install again within the application.

The following are some useful references for working with ESLint:

- For instructions on how to work with ESLint, see `https://eslint.org/docs/user-guide/`.
- For details on how ESLint works, see `https://eslint.org/docs/developer-guide/`.

Downloading and Installing LiveServer

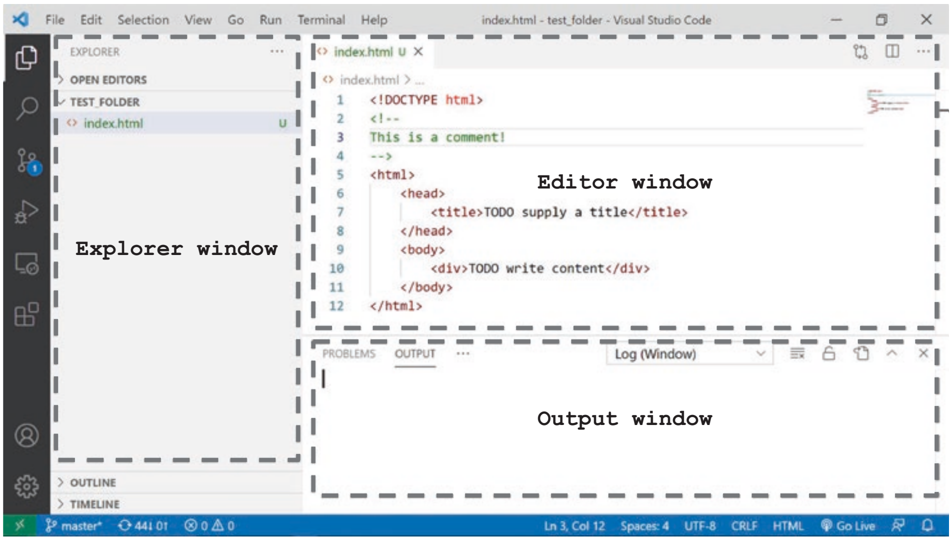
The LiveServer extension to the VS Code is required to run your game engine. It launches a web server locally on your computer through VS Code to host developed games. Much like ESLint, you can install LiveServer with the following steps:

- Go to `https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer` and click install.
- You will be prompted to open VS Code and may need to click install again within the application.

Working in the VS Code Development Environment

The VS Code IDE is easy to work with, and the projects in this book require only the editor. Relevant source code files organized under a parent folder are interpreted by VS Code as a project. To open a project, select File ➤ Open Folder and navigate and select the parent folder that contains the source code files of the project. Once a project is open, you need to become familiar with the basic windows of VS Code, as illustrated in Figure 1-1.

- **Explorer window**: This window displays the source code files of the project. If you accidentally close this window, you can recall it by selecting View ➤ Explorer.
- **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 clicking once the corresponding file name in the Explorer window.
- **Output window**: This window is not used in our projects; feel free to close it by clicking the “x” icon on the top right of the window.



```
<!DOCTYPE html>
<!--
This is a comment!
-->
<html>
    <head>
        <title>TODO supply a title</title>
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

Figure 1-1. *The VS Code IDE*

Creating an HTML5 Project in VS Code

You are now ready to create your first HTML5 project:

- Using File Explorer, create a directory in the location where you would like to keep your projects. This directory will contain all source code files related to your projects. In VS Code, select File ➤ Open Folder and navigate to the directory you created.

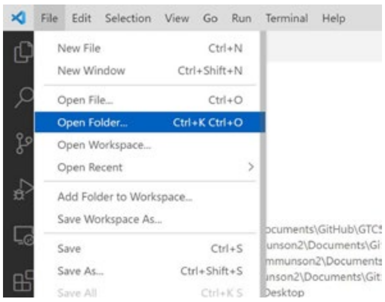


Figure 1-2. *Opening a project folder*

- VS Code will open the project folder. Your IDE should look similar to Figure 1-3; notice that the Explorer window is empty when your project folder is empty.

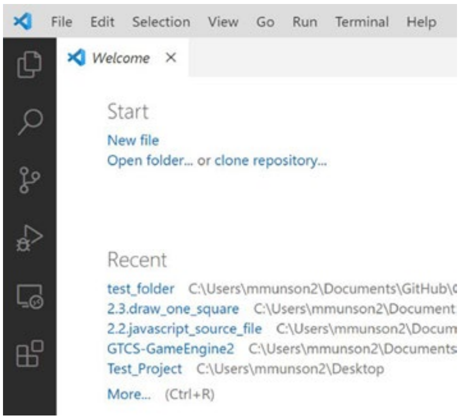

File Edit Selection View Go Run Terminal Help
Welcome
Start
New file
Open folder... or clone repository...
Recent
test_folder C:\Users\mmunson2\Documents\GitHub\
2.3.draw_one_square C:\Users\mmunson2\Document
2.2.javascript_source_file C:\Users\mmunson2\Docum
GTCS-GameEngine2 C:\Users\mmunson2\Documents
Test_Project C:\Users\mmunson2\Desktop
More... (Ctrl+R)

Figure 1-3. *An empty VS Code project*

- You can now create your first HTML file, `index.html`. Select File ➤ New File and name the file `index.html`. This will serve as the home or landing page when your application is launched.

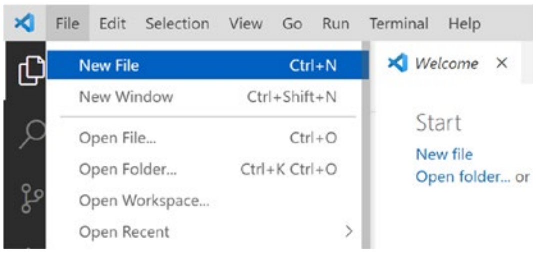


Figure 1-4. *Creating the `index.html` file*

- In the Editor window, enter the following text into your `index.html`:

```
<!DOCTYPE html>
<!--
This is a comment!
-->
<html>
    <head>
        <title>TODO supply a title</title>
    </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.

As illustrated in Figure 1-5, you can run this project by clicking the "Go Live" button in the bottom-right corner of your VS Code or by pressing Alt+L Alt+O. There is a chance that right after you entered the previous HTML code for the first time, the "Go Live" button may not appear. In this case, simply right-click the `index.html` file in the Explorer window, and click "Open with Live Server" menu item to launch the web page. After the first time, the "Go Live" button will appear in the lower-right region of the IDE, as illustrated in Figure 1-5.

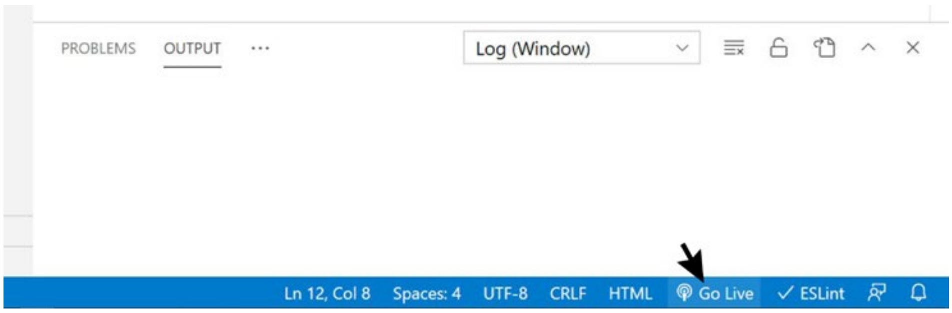


Figure 1-5. *Click the Go Live button to run a project*

Note To run a project, the `index.html` file of that project must be opened in the editor when the "Go Live" button is clicked or when the Alt+L Alt+O keys are typed. This will become important in the subsequent chapters when there are other JavaScript source code files in the project.

Figure 1-6 shows an example of what the default project looks like when you run it. Notice that after the project begins to run, the "Go Live" button updates its label to show "Port:5500." You can click this button again to disconnect the IDE from the web page to observe the "Go Live" label again. Clicking the button one more time will rerun the project.