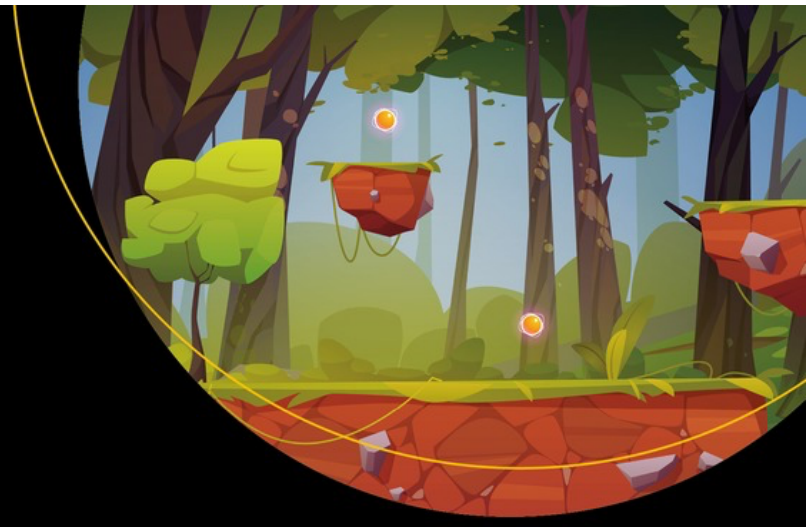

Build Your Own 2D Game Engine and Create Great Web Games

Using HTML5, JavaScript, and WebGL2

—

Second Edition

—

Kelvin Sung
Jebediah Pavleas
Matthew Munson
Jason Pace

Apress®

Introducing 2D Game Engine Development

The goal of this series of lessons is to build a 2D game engine. This process will teach students how game engines work. The lessons will use a number of technologies to work. Students do need programming experience but not necessarily in the languages and technologies. The following list offers links for obtaining additional information on technologies used in this book:

- JavaScript: www.w3schools.com/js
- HTML5: www.w3schools.com/html/html5_intro.asp
- WebGL: www.khronos.org/webgl
- WebGL Tutorial: https://webgl2fundamentals.org
- OpenGL: http://www.opengl.org
- Visual Studio Code: https://code.visualstudio.com/
- Chrome: www.google.com/chrome
- glMatrix: glMatrix.net
- ESLint: www.eslint.org

Over the course of this text, other recommendations will come up. These are optional but helpful additions to the tools used above.

Book Conventions

In this book, you will encounter a number of formatting conventions. Each section of a chapter will be **centered and BOLD** while informational sections will be left-justified and **bold**. If you are at a section where you will need to do something in code, the header will be in *italic and highlighted green*. Each chapter consists of one or more labs. You will first be introduced to concepts and then you will implement the concepts in code. Code is provided in the lab sections. As you progress through the projects, you will make a copy of the end result of the lab to serve as a starting off point for the next lab. If you want to experiment, you should always make a copy of a lab.

Goals will appear in peach-colored boxes.

Learn about 2D game engines

A **TASK** appears in an orange box.

TASK write a Java program that writes a greeting.

A tip/reminder will be presented in a white box.

When you see a tip, it is usually information about a newly introduced concept.

A **NOTE** will be found in a yellow block that can provide extra information beyond the scope of the discussion about the current topic or a reference on where you can get more information.

Note you can find out more about this at http://xyz.com

A `code block` will be provided in a blue box using a monospace font. This code is meant to be copied directly into your project.

```
public static void main() {
   System.out.println("Hello World!");
}
```

Sometimes code will have sections in **bold**. This can either indicate code that you need to add to previous code…

```
public static void main() {
   System.out.println("Hello World!");
   System.out.println("Have a Great Day!");
}
```

Or **bold** can represent code that is to be retained from previously added code which serves as a placeholder to help you determine where to add new code.

```
public static void main() {
   ... identical to previous code ...
   System.out.println("And don't forget your jacket when you leave.");
}
```

Elements to Game Design

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. These lessons are 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 (e.g., 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 specific implementation of that mechanic is an aspect of systems design, level design, and the interaction model/game loop.

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. Systems that appear fairly simple to players may require many components working together and balanced perfectly against each other, and underestimating system 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 combines 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). 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 a good practice to think of these two elements as inseparable. 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 (e.g., 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. Game settings need not be elaborate to be effective; the perennially popular puzzle game *Tetris* has a rather simple setting with no real narrative wrapper, but the combination of abstract setting, audiovisual design, and level design is uniquely well matched and contributes significantly to the millions of hours players invest in the experience year after year.

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. 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.

Prep 1

The purpose of this lab is set up the environment for programming the 2D engine. The IDE these lessons are based is Microsoft Visual Studio Code. To work with VS Code, you will need to install the IDE and a number of supporting elements.

1. Install the IDE. This can be found at https://code.visualstudio.com/
2. Download the glMatrix math library found at http://glMatrix.net/ (you just need to download it… we will work with it at a later time)
3. Install LiveServer found at https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer
4. (**Optional**) You can install ESLint. This is a tool used with JavaScript to locate code style inconsistencies and coding quality issues. There is significant configuration steps required to use ESLint and these steps go beyond the scope of this text. Download at https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint (you view information on configuration at https://www.youtube.com/watch?v=SYSZi_nQzxk)

Prep 2

The purpose of this lab is to become familiar with VS Code. 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 below.

- **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.

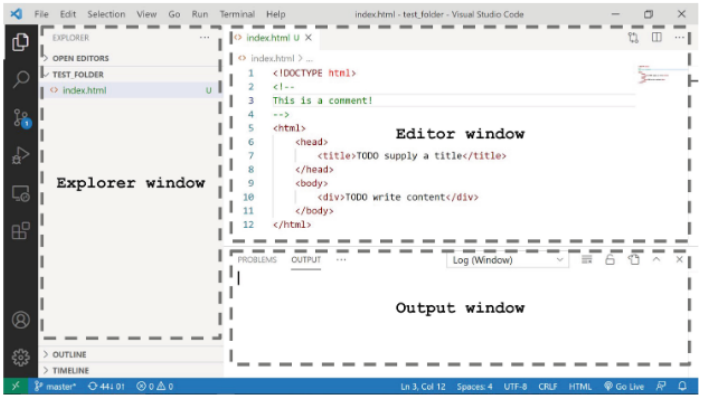

Figure 1-1 The Visual Studio Code window

Lab 1.1

The purpose of this lab is to create an environment for your game engine.

1. Create an HTML5 project in VS Code called `CH1.1`
   - Create a new folder for your project in the **Finder**
   - In VS Code, go to the **File** menu and select **Open Folder…**
   - Navigate to the folder you created earlier and select **Open**
2. Create an HTML file.
   - Go to the **File** menu and select **New File**
   - Name the file `index.html`
3. In the Editor window, enter the following in the `index.html` file.

```
<!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 make 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.

4. View the web page in the Google Chrome browser. To do this, VS Code needs to run a webserver. This can be done by clicking on the **GoLive** button in the bottom-right of the Editor window

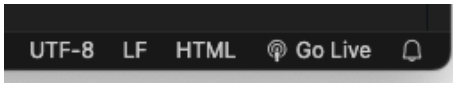

Figure 1-2 The "Go Live" button

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. This will become important in the subsequent chapters when there are other JavaScript source code files in the project.

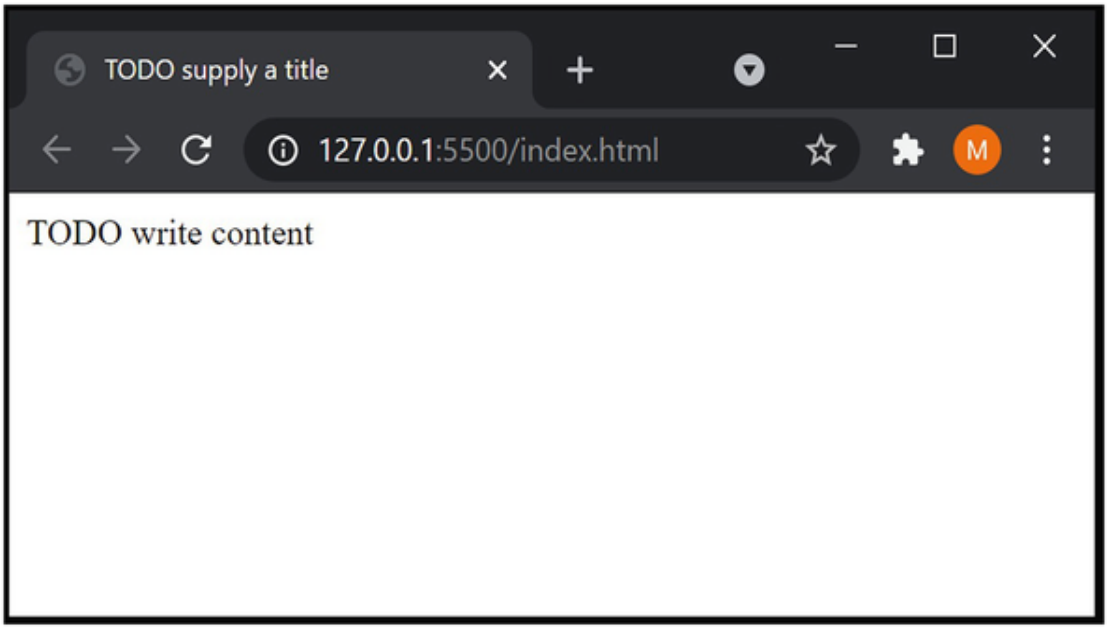

Figure 1-3 The Final Product