# Introduction

## The Great Game of Programming

I have a firmly held personal belief, grown from decades of programming: in a very real sense, programming is a game. At least, it can be *like* playing a game with the right mindset.

For me, spending a few hours programming—crafting code that bends these amazing computational devices to my will and creating worlds of living software—is entertaining and rewarding. It competes with delving into the Nether in *Minecraft*, snatching the last Province card in *Dominion*, or taking down a reaper in *Mass Effect*.

I don't mean that programming is mindless entertainment. It is rarely that. Most of your time is spent puzzling out the right next step or figuring out why things aren't working as you expected. But part of what makes games engaging is that they are challenging. You have to work for it. You apply creativity and explore possibilities. You practice and gain abilities that help you win.

You'll be in good shape if you approach programming with this same mindset because programming requires this same set of skills. Some days, it will feel like you are playing *Flappy Bird*, *Super Meat Boy,* or *Dark Souls*—all notoriously difficult games—but creating software is challenging in all the *right* ways.

The "game" of programming is a massively multiplayer, open-world sandbox game with role-playing elements. By that, I mean:

- **Massively multiplayer:** While you may tackle specific problems independently, you are never alone. Most programmers are quick to share their knowledge and experience with others. This book and the Internet ensure you are not alone in your journey.
- **An open-world sandbox game:** You have few constraints or limitations; you can build what, when, and how you want.
- **Role-playing elements:** With practice, learning, and experience, you get better in the skills and tools you work with, going from a lowly Level 1 beginner to a master, sharpening your skills and abilities as you go.

If programming is to be fun or rewarding, then learning to program must also be so. Rare is the book that can make learning complex technical topics anything more than tedious. This book attempts to do just that. If a spoonful of sugar can help the medicine go down, then there

must be some blend of eleven herbs and spices that will make even the most complex technical topic have an element of fun, challenge, and reward.

Over the years, strategy guides, player handbooks, and player's guides have been made for popular games. These guides help players learn and understand the game world and the challenges they will encounter. They provide time-saving tips and tricks and help prevent players from getting stuck anywhere for too long. This book attempts to be that player's guide for the Great Game of Programming in C#.

This book skips the typical business-centric examples found in other books in favor of samples with a little more spice. Many are game-related, and many of the hands-on challenges involve building small games or slices of games. This makes the journey more entertaining and exciting. While C# is an excellent language for game development, this book is not specifically a C# game programming book. You will undoubtedly come away with ideas to try if that's the path you choose, but this book is focused on becoming skilled with the C# language so that you can use it to build *any* type of program, not just games. (Most professional programmers make business-centric applications, web apps, and smartphone apps.)

This book focuses on console applications. Console applications are those text-based programs where the computer receives text input from the user and displays text responses in the stereotypical white text on a black background window. We'll learn some things to make console applications more colorful and exciting, but console applications are, admittedly, not the most exciting type of application.

Why not use something more exciting? The main reason is that regardless of whether you want to build games, smartphone apps, web apps, or desktop apps, the *starting points* in those worlds already expect you to know much about C#. For example, I just looked over the starter code for a certain C# game development framework. It demands you already know how to use advanced topics covered in Level 25 (inheritance), Level 26 (polymorphism), and Level 30 (generics) just to get started! While some people successfully dive in and stay afloat, it is usually wiser to build up your swimming skills in a lap pool before trying to swim across the raging ocean. Starting from the basics gives you a better foundation. After building this foundation, learning how to make specific application types will go much more smoothly. Few will be satisfied with just console applications, but spending a few weeks covering the basics before moving on will make the learning process far easier.

## Book Features

Creating a fun and rewarding book (or at least not a dull and useless one) means adding some features that most programming books do not have. Let's look at a few of these so that you know what to expect.

### Speedruns

At the start of each level (chapter) is a Speedrun section that outlines the key points described in the level. It is not a substitute for going through the whole level in detail but is helpful in a handful of situations:

1. You're reviewing the material and want a reminder of the key points.
2. You are skimming to see if some level has information that you will need soon.
3. You are trying to remember which level covered some particular topic.

### Challenges and Boss Battles

Scattered throughout the book are hands-on challenges that give you a specific problem to work on. These start small early in the book, but some of the later ones are quite large. Each of these challenges is marked with the following icon:

When a challenge is especially tough, it is upgraded to a Boss Battle, shown by the icon below:

Boss Battles are sometimes split across multiple parts to allow you to work through them one step at a time.

I strongly recommend that you do these challenges. You don't beat a game by reading the player's guide. You don't learn to program by reading a book. You will only truly learn if you sit down and program.

I also recommend you do these challenges as you encounter them instead of reading ten chapters and returning to them. The *read a little, program a little* model is far better at helping you learn fast.

I also feel that these challenges should not be the *only* things you program as you learn, especially if you are relatively new to programming. Half of your programming time should come from these challenges and the rest from your own imagination. Working on things of your own invention will be more exciting to you. But also, when you are in that creative mindset, you mentally explore the programming world better. You start to think about how you can apply the tools you have learned in new situations, rather than being told, "Go use this tool over here in this specific way."

As you do that, keep in mind the size of the challenges you are inventing for yourself. If you are learning how to draw, you don't go find millennia-old chapel ceilings to paint (or at least you don't expect it to turn out like the Sistine Chapel). Aim for things that push your limits a little but aren't intimidating. Keep in mind that everything is a bit harder than you initially expect. And don't be afraid to dive in and make a mess. Continuing the art analogy, you aren't learning if you don't have a few garbage drawings in your sketchbook. Not every line of code you write will be a masterpiece. You have permission to write strange, ugly, and awkward code.

If these specific challenges are not your style, then skip them. But substitute them with something else. You will learn little if you don't sit down and write some code.

When a challenge contains a **Hint**, these are suggestions or possibilities, not things you must do. If you find a different path that works, go for it.

Some challenges also include things labeled **Answer this question**. I recommend writing out your answer. (Comments, covered in Level 4, could be a good approach.) Our brains like to tell us it understands something without proving it does. We mentally skip the proof, often to our detriment. Writing it out ensures we know it. These questions usually only take a few seconds to answer.

I have posted my answers to these challenges on the book's website, described later in this introduction. If you want a hint or compare answers, you can review what I did. Just because our solutions are different doesn't make yours bad or wrong. I make plenty of my own

mistakes, have my own preferences for the various tools in the language, and have also been programming in C# for a long time. As long as you have a working solution, you're doing fine.

### Knowledge Checks

Some levels in this book focus on conceptual topics that are not well-tested by a programming problem. In these cases, instead of a Challenge problem, these levels will have a Knowledge Check, containing a quiz with true/false, multiple-choice, and short answer questions. The answers are immediately below the Knowledge Check, so you can see if you learned the key points right away. These are marked with the knowledge scroll icon below:

### Experience Points and Levels

Since this book is a player's guide, I've attempted to turn the learning process into a game. Each Challenge and Knowledge Check section is marked in the top right with experience points (written as *XP*, as most games do) that you earn by completing the challenge. When you complete a challenge, you can claim the XP associated with it and add it to your total. Towards the front of this book, after the title page and the map, is an XP Tracker. You can use this to track your progress, check off challenges as you complete them, and mark off your progress as you go.

You can also get extra copies of the XP Tracker on the book's website (described below) if you do not want to write in your book, have a digital copy, or have a used copy where somebody else has already marked it.

As you collect XP, you will accumulate enough points to level up from Level 1 to Level 10. If you reach Level 10, you will have completed nearly every challenge in this book and should have an excellent grasp of C# programming.

The XP assigned to each challenge is not random. Easier challenges have fewer points; more demanding challenges are worth more XP. While measuring difficulty is somewhat subjective, you can generally count on spending more time on challenges with more points and will gain a greater reward for it.

### Narratives and the Plot

The challenges form a loose storyline that has you, the (soon to be) Master Programmer journeying through a land that has been stripped of the ability to program by the malevolent, shadowy, and amorphous Uncoded One. Using your growing C# programming skills, you will be able to help the land's inhabitants, fend off the Uncoded One's onslaught, and eventually face the Uncoded One in a final battle at the end of the book.

Even if this plot is not attractive to you, the challenges are still worth doing. Feel free to ignore the book-long storytelling if it isn't helpful for you.

While much of the book's "plot" is revealed in the Challenge descriptions themselves, there were places where it felt shoehorned. Narrative sections supplement the descriptions in the challenges but otherwise have no purpose beyond advancing this book-long plot. These are marked with the icon below:

If you are ignoring the plot, you can skip these sections. They do not contain information that helps you be a better C# programmer.

### Side Quests

While everything in this book is worth knowing (skilled C# programmers know all of it), some sections are more important than others. Sections that may be skipped in your first pass through this book are marked as Side Quests, indicated with the following icon:


These often deal with situations that are less common or less impactful. If you're pressed for time, these sections are better to skip than the rest. However, I recommend returning to them later if you don't read them the first time around.

### Glossary

Programmers have a mountain of unique jargon and terminology. Beyond learning a new programming language, understanding this jargon is a second massive challenge for new programmers. To help you with this undertaking, I have carefully defined new concepts within the book as they arise and collected all of these new words and concepts into a glossary at the back of the book. Only the lucky few will remember all such words from seeing it defined once. Use the glossary to refresh your mind on any term you don't remember well.

### The Website

This book has a website associated with it, which has a lot of supporting content: **https://csharpplayersguide.com**. Some of the highlights are below:

- **https://csharpplayersguide.com/solutions**. Contains my solutions to all the Challenge sections in this book. My answer is not necessarily more correct than yours, but it can give you some thoughts on a different way to solve the problem and perhaps some hints on how to progress if you are stuck. This also contains more thorough explanations for all of the Knowledge Checks in the book.
- **https://csharpplayersguide.com/errata**. This page contains errata (errors in the book) that have been reported to clarify what was meant. If you notice something that seems wrong or inconsistent, you may find a correction here.
- **http://csharpplayersguide.com/articles**. This page contains a collection of articles that add to this book's content. They often cover more in-depth information beyond what I felt is reasonable to include in this book or answer questions readers have asked me. In a few places in this book, I call out articles with more information for the curious.

### Discord

This book has an active Discord server where you can interact with me and other readers to discuss the book, ask questions, report problems, and get feedback on your solutions to the challenges. Go to **https://csharpplayersguide.com/discord** to see how to join the server. This server is a guildhall where you can rest from your travels and discuss C# with others on a similar journey as you.

## I Want Your Feedback

I depend on readers like you to help me see how to make the book better. This book is much better because past readers helped me know what parts were good and bad.

Naturally, I'd love to hear that you loved the book. But I need constructive criticism too. If there is a challenge that was too hard, a typo you found, a section that wasn't clear, or even that you felt an entire level or the whole book was bad, I want to hear it. I have gone to great lengths to make this book as good as possible, but with your help, I can make it even better for those who follow in our footsteps. Don't hesitate to reach out to me, whether your feedback is positive or negative!

I have many ways that you can reach out to me. Go to **https://csharpplayersguide.com/contact** to find a way that works for you.

## An Overview

Let's take a peek at what lies ahead. This book has five major parts:

- **Part 1—The Basics.** This first part covers many of the fundamental elements of C# programming. It focuses on procedural programming, including storing data, picking and choosing which lines of code to run, and creating reusable chunks of code.
- **Part 2—Object-Oriented Programming.** C# uses an approach called object-oriented programming to help you break down a large program into smaller pieces that are each responsible for a little slice of the whole program. These tools are essential as you begin building bigger programs.
- **Part 3—Advanced Topics.** While Parts 1 and 2 deal with the most critical elements of the C# language, there are various other language features that are worth knowing. This part consists of mostly independent topics. You can jump around and learn the ones you feel are most important to you (or skip them all entirely, for a while). In some ways, you could consider all of Part 3 to be a big Side Quest, though you will be missing out on some cool C# features if you skip it all.
- **Part 4—The Endgame.** While hands-on challenges are scattered throughout the book, Part 4 consists of a single, extensive, final program that will test the knowledge and skills that you have learned. It will also wrap up the book, pointing you toward Lands Uncharted and where you might go after finishing this book.
- **Part 5—Bonus Levels.** The end of the book contains a few bonus levels that guide you on what to do when you don't know what else to do—dealing with compiler errors and debugging your code. The glossary and index are also back here at the end of the book.

Please do not feel like you must read this book cover to cover to get value from it.

If you are new to programming, I recommend a slow, careful pace through Parts 1 and 2, skipping the Side Quests and only advancing when you feel comfortable taking the next step. After Part 2, you might continue your course through the advanced features of Part 3, or you might also choose to skim it to get a flavor for what else C# offers without going into depth. Even if you skim or skip Part 3, you can still attempt the Final Battle in Part 4. If you're making consistent progress and getting good practice, it doesn't matter if your progress feels slow.

If you are an experienced programmer, you will likely be able to race through Part 1, slow down only a little in Part 2 as you learn how C# does object-oriented programming, and then spend most of your time in Part 3, learning the things that make C# unique.

Adapt the journey however you see fit. It is your book and your adventure!