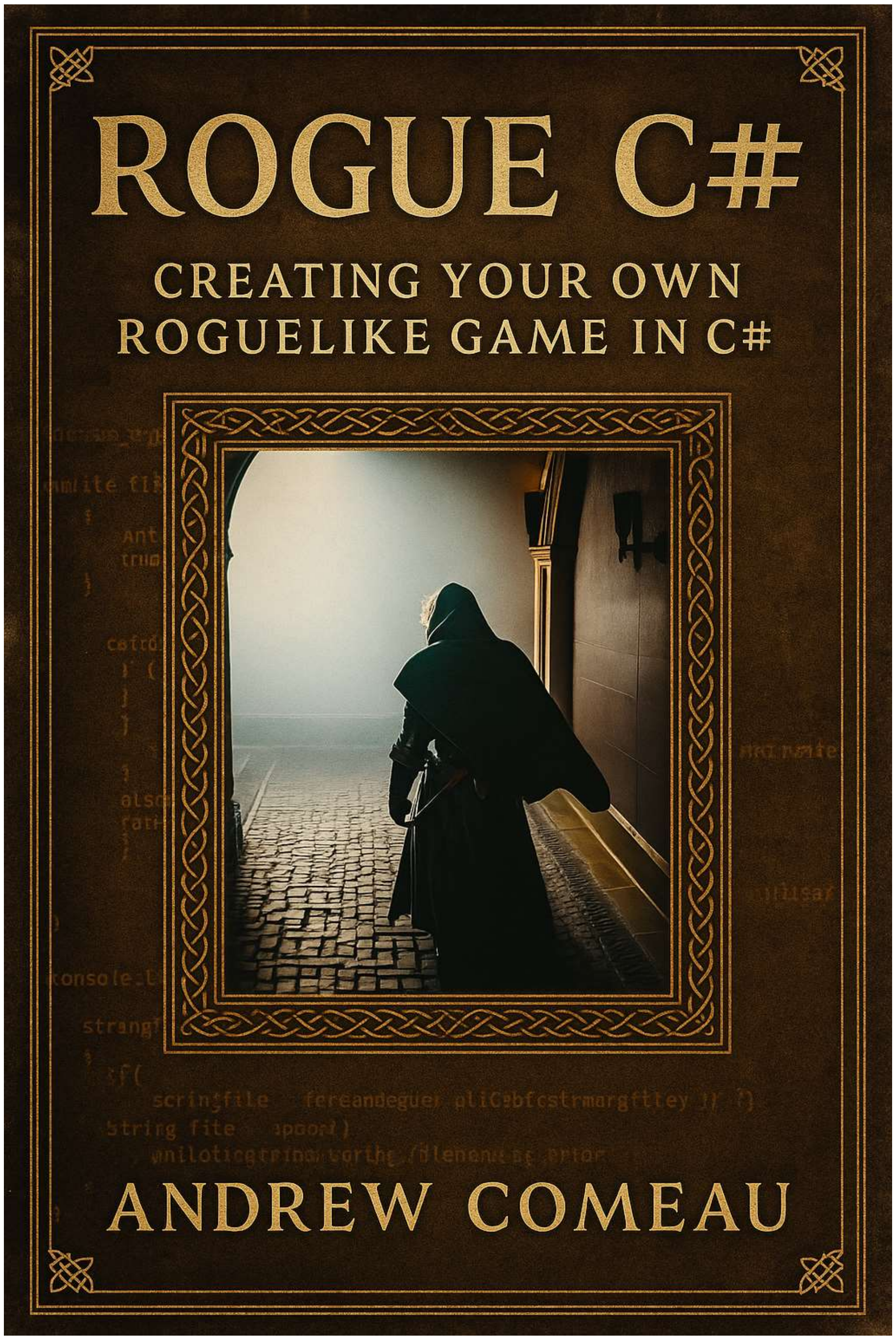


Content adapted from the web series on:

ComeauSoftware.com

Updated June 2025

Table of Contents

Introduction

A few years ago, when I was teaching a college course in database programming, I remember one of the students looking at a sample program and wondering out loud "How does anyone actually put something like this together?". I knew what he was asking - it's a long way from playing around with some code snippets to actually organizing and building an application from the ground up. It can be as much work as writing a novel or building a work shed or any other major project.

I don't say any of this to discourage anyone from programming. I've been doing it off and on since I was a teenager. Even as a student, I was coming up with ideas of my own for projects and hammering away at them for hours after school. It was often frustrating but I kept doing it because the high that comes when something finally works like you want it to feels like nothing else.

By the time I was teaching, I'd decided that the best way to learn programming was mainly through practice. Systematic study of a language is great to get a handle on the syntax but it's a waste of time without the continued practice that will commit the ideas to memory. The class I taught ended up being about 90% workshop as I threw one challenge after another at the students and made them research solutions on their own. Software development is not just about knowing the language elements; it's about breaking problems down into pieces and learning to use the programming tools to create a solution. That only comes with experience, research and lots of practice.

So, for this demo, I'm accepting a single, epic challenge - the rebuilding of a classic role-playing game called ... **Rogue**.

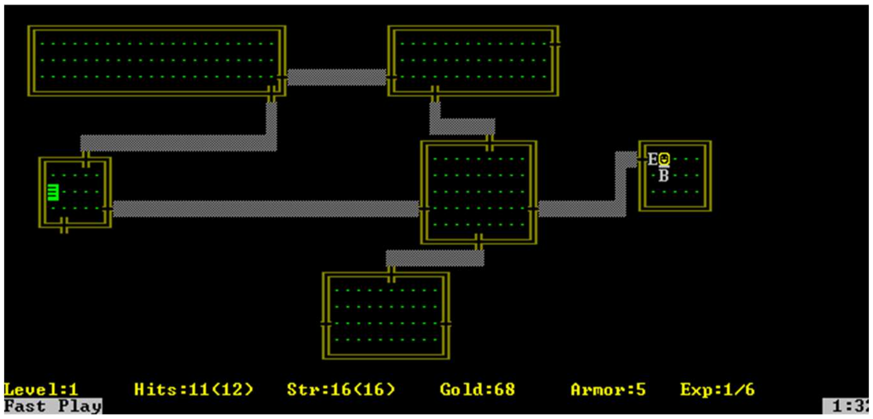
"Oh, yeah, Rogue! ... Wait, what???"

Rogue is a role-playing game (RPG) that was first released in 1980 for Unix-based systems and then ported to other operating systems, including the up-and-coming MS-DOS. It's a 2-D game where your character wanders around dungeons fighting monsters, picking up gold and other items and trying to find the Amulet of Yendor so you can escape the dungeon and win the game.

Rogue is turn-based so every time you move, so do the monsters. If your character dies then you start the game over, the graphics are entirely made up of text characters and it doesn't look much like the sophisticated games of today.

Rogue is also a classic game that inspired an entire genre of other roguelike games and even a set of standards by which to measure these games. Designing a Rogue clone is a complex programming challenge involving graphics design, game logic, the management of many elements and the consistent display of those elements on screen.

As a game, Rogue is incredibly addictive. Dennis Ritchie, the creator of the C programming language, joked that it was "the biggest waste of CPU cycles in history". Jerry Pournelle, scientist and sci-fi author, called it his "game of the month". In 2009, it was #6 on PC World's list of the ten greatest PC games ever. I personally deleted all my copies of it at one point just to stay away from it.



```
Level:1      Hits:11(12)   Str:16(16)    Gold:68      Armor:5    Exp:1/6
Fast Play                                                                  1:3
```

Since this is an RPG with character stats and "dice" rolls, you'll see how to program those calculations and apply them within the game. Every turn involves multiple decisions to update the game board and character stats. You'll also learn how to load and save game data.

Despite the retro appearance of the game, every skill mentioned above is still essential to modern gaming and software development. The lack of modern graphics will let you focus on learning and applying the actual C# language and concepts to design your own application. During this series, you'll also learn something about software requirements, testing and troubleshooting - important skills in a software development career.

Rogue was originally developed in C as, essentially, a console application. As of Windows 10, the original game is no longer playable on Windows unless you use a virtual machine but you can get a version from Steam that runs through DOSBox. For this course, I'm re-creating the game using a Windows Forms application while trying to keep as much of the original look and feel as possible.

Of course, cloning a game doesn't mean that I can't throw in a few new features. Every Rogue clone that's been designed over the years has added its own style and there are a couple of things that I'd like to change. The original Rogue didn't have any options for the user to customize and I'd like to add a couple. Hopefully, by the end of this course, you'll be able to program in your own features.

Who is this series for?

I've tried to make these articles as friendly for newcomers as possible but it's not a language reference. There are already plenty of online tutorials and references that do a fine job of teaching C# basics. W3Schools.com has great references for C# and many other languages that will present the language in bite-size chunks and let you practice it right on their site. I'll frequently reference their site and others if there's something you need to be familiar with when going through one of the chapters in this course.

The idea behind this series is to learn how to actually *use* the language to build a fully-featured application using the best practices and techniques available to you. When I do talk about the basics, it will be in the context of building the application. Along the way, you will see many concepts and language features demonstrated and will hopefully get a better grasp on how to use the different parts of the language. Hopefully, over the course of this series, you'll see the overall strategy and organization involved in planning and building a piece of software.

When I reference another programming resource, I strongly suggest you check it out and bookmark the site. Even as an experienced programmer, I frequently turn to Google for answers on how to use specific functions. With a language as large as C#, being able to research and find information is actually more important than memorizing the language.

If you get a couple chapters in and find you need something more basic first, check out the following and then come back and try again:

C# Fundamentals for Absolute Beginners
(https://learn.microsoft.com/en-us/shows/csharp-fundamentals-for-absolute-beginners/)

Wise Owl series on YouTube
(https://www.youtube.com/playlist?list=PLanFskMAuuGduNf07DthlWaix6WT4rUMD)

What you need for this series

I'm developing this demo right here on ComeauSoftware.com where you'll be able to follow it for free as I release new chapters. I originally envisioned it as a project where readers could code along with me and develop their own roguelike from the ground up. That proved to be impractical because of the amount of code involved and the ongoing development of the game. A single change or new feature can affect enough points in the project that expecting you as the reader to recreate all the changes would be unrealistic.

To get the most out of Rogue C#, you should first download the current source code from Github where I've made it available for your reference. You can also view it directly on the site if you would rather not download it.

https://github.com/ajcomeau/RogueGameDev

If you're not familiar with Github, it really is one of the first things you should learn as an aspiring software developer. In simplest terms, Git is a gigantic repository for source code and software projects. Developers of all kinds can upload their source code, track changes and allow other developers to share in the development effort. The *Github* website is the online graphical interface for the Git service. Check out the video below for more of an explanation.

https://youtu.be/2ReR1YJrNOM

To use the code, you should have already installed Visual Studio 2022 Community Edition or higher. The Community edition is a free installation from Microsoft and I'm using it myself so it has everything you'll need to follow along.

Installing Visual Studio
https://learn.microsoft.com/en-us/shows/csharp-fundamentals-for-absolute-beginners/installing-visual-studio

After that, all you need is the time and commitment to read through the chapters and try things out for yourself. The first part of the course deals with the construction of the game map and, as far as possible, I provide step-by-step instructions on how to apply the settings and enter the code. There is *a lot* of code, however, and the later chapters tend to focus on explaining the algorithms and decision processes behind the development.

I still recommend that you create your own version of the game and follow along as much as possible. It's also good to have an extra "sandbox" project where you can try things out without disrupting the main demo project. Every new function or code element you see is another tool that you can learn to use. You should pay close attention to the code samples and the explanations behind them and think of ways that you can try them on your own.

What this series is *not* ...

This series will help you learn the C# programming language using a single, large game project. Unlike a formal course, it will demonstrate language features as they can be applied in the project and not in a specific order.

- **This is a demonstration, not a systematic tutorial.** I will present the basic parts of programming that every beginner should learn, either through the main project or as side explanations. I cannot promise that I will address *every* subject in C# that you might expect or to give them equal time.
- Unless otherwise stated, **the programming solutions I show are specific to this project and are not necessarily the best in *all* situations**. This is especially true as I am replicating the look and feel of a 40-year old PC console game in a modern Windows Forms application. Any given challenge can be solved in a variety of ways with a language as rich and varied as C# and I emphasize my own solutions that will best accomplish the goals of the program.
- Because I am developing the game and writing the chapters of this demo at the same time, **It's entirely possible that I will discover that something I did earlier in the series doesn't work.** Depending on the problem and its usefulness as a lesson, I will either go back and change the earlier chapter(s) or I will write it up as a change. If you find something in here that doesn't work, please let me know.
- The game developed in this project is inspired by the original Rogue DOS game but I don't guarantee it will fit any specific "roguelike" or "roguelite" standard.

My goal in this series is to demonstrate the C# language and programming in a way that will enable you to carry on with your own projects afterward ... and have fun doing it!

Notes on the PDF Edition

The PDF version of this series that you're reading is a quick conversion of the original web articles to PDF with no extra content added. This is simply a bonus feature of the series for those who might like all the content in this portable format.

As of June 2025, I am reviewing the series with an eye toward finishing it in book form or at least bringing it to some kind of conclusion. I recommend that you bookmark my site at ComeauSoftware.com to keep up to date on any new developments.

With that in mind, let's get started.

Andrew Comeau
Comeau Software Solutions
June 2025

Choosing a Programming Language

The choice of programming language and other technologies is one of the first steps in building software. With the variety of languages and the overlap between them, the choice often depends on the preferred environment (i.e. desktop vs. web), operating system and personal preferences of the developer or company. For the individual programmer, the choice of a first programming language can be confusing but is ultimately less important than the continued dedication to developing your skills and knowledge of overall programming concepts with whatever language you choose.

What language should I learn?

That's the first question that I often see from aspiring programmers which is understandable. With all the technologies out there, it's easy to get lost in all the choices and spend too much time agonizing over the decision and second-guessing yourself.

There are **two** basic questions you should ask yourself when deciding on a first language –

1. What kind of programming do I want to do?
2. What language am I most comfortable with?

That's really what it comes down to. I'm not going to defend C# as the *best* first language to learn because it might not *be* the best for any given person. It's simply the language that I find most useful at this point and the one I'm choosing to teach here. If you're looking at this course, you probably have more than a passing interest in learning it, too, and I'll do the best I can to guide you.

What do you want to create?

To answer the first question, C# is a general purpose language derived from C++ as so many languages are. As a general purpose language, I've used it to code everything from Windows Forms apps to web applications with ASP.NET to Windows services. That's something I like about it as I have one major language that I can do just about anything with. I've also seen the tendency for people, including myself, to get distracted by every new popular programming technology that comes along and to believe that they have to know a lot of different languages in order to be real programmers. This is not true.

Of course, if your main language is no longer in demand and you want to stay in the job market, it's a good idea to pick up something new and popular to stay in the game. As long as you're working with useful tools, though, there's nothing wrong with just mastering those tools as thoroughly as possible. Certainly, there's more than enough in a language like C# to keep you busy doing that, especially as it's now being updated every year.

It's great to have *some* experience with two or three languages for the sake of perspective but, if you really enjoy programming, you'll probably pick that up naturally as you go. You don't need to be fluent in a variety of human languages to communicate with others and you don't need to know a variety of computer languages to call yourself a programmer.

What language are you comfortable with?

If I was choosing a language just based on popularity and marketability, I might go with JavaScript but I don't really *enjoy* working with JavaScript. I've played with it here and there and even used it for a couple of projects so I'm confident that I can learn it as I need to but I've never really needed to. I personally prefer a compiled language over an interpreted browser language and the strong typing and strict structure of C# over the more casual attitude of JavaScript. I like the development tools for C# better and Microsoft's Visual Studio has come so far that it's just fun to use.

Again, it never hurts to investigate a new language because you just might find that you really like it. I was a Visual Basic programmer until I found myself on a team that used C#. I was productive with it within a couple of weeks and have never gone back although I still enjoy working with Visual Basic for Applications (VBA) when I need to.

As a programmer, you'll spend enough time and energy mastering various concepts such as object orientation and data management no matter what language you use. As long as you're reaching your personal goals and creating great things, forcing yourself to learn a different language for any other reason is often a distraction and will take the fun out of programming.

Above all, use a language that will help you *keep programming*.

What else should I learn? What will I need to know tomorrow?

C# is not going anywhere. I'm confident that the language will be around for many years to come and it's used enough that, even after something else takes its place, there will be plenty of code that needs to be maintained. Still, there are other concepts such as user interface standards and error handling that apply no matter what technology you're using and I'll do my best to present as many of these as I can during this course.

C# is the chosen language in this course and I want you to be comfortable with it by the end but I also want you to know how to design great, robust solutions. While there are guidelines that you can follow, there is no checklist. It really comes down to understanding how the technology works and caring about the outcome so that you can make the right decisions at each point in a project.

Exploring Further

Throughout this series, I will encourage you to explore certain subjects further on your own. I'll be sharing links and encouraging you to do your own searches on specific subjects. This is an essential skill for programmers but, unfortunately, its one that aspiring programmers sometimes lack. So many people will spend tens of thousands of dollars on bootcamps and classroom instruction when there are endless resources of all types online, even free college courses. Technology is constantly evolving and finding a new classroom course or bootcamp for every new subject would soon become impractical.

I encourage you to pay close attention to the external links that I feature throughout the course. I enjoy sharing quality resources and I will do that here as well. Take a moment to bookmark some of the sites that I refer to as they might come in very handy as you progress in your own programming journey.

JavaScript introduction at developer.mozilla.org
https://developer.mozilla.org/en-US/docs/Web/JavaScript

FreeCodeCamp.org – Interpreted vs. Compiled Languages
https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/

Harvard University – CS50x: Introduction to Computer Science through OpenCourseWare
https://cs50.harvard.edu/x/2022/

See the online version of this chapter with links to more resources here:
https://www.comeausoftware.com/c-sharp/rogue-csharp-choosing-a-language/

Github repository:
https://github.com/ajcomeau/RogueGameDev

Writing the Program Requirements

Requirements analysis and writing lays the foundation for building a good piece of software. The requirements defined for a program form the outline on which the rest of the project can be built. It's usually a cooperative and ongoing process involving many conversations between the developers and intended users. Requirements documentation can take different forms depending on the intended audience and use.

While the requirements for this project are demonstrated through the original Rogue game, it is still important to review the expected performance of the new game. This gives a general idea of how the application will need to be organized in order to perform as needed while staying maintainable and open to later expansion. This chapter finishes by looking at some of the most basic features of the original console game from 1980 and how they'll translate into a modern Windows application.

Introduction

One of the most important tasks in software development is defining the software *requirements* – in detail. Architects have blueprints, writers develop outlines and smart programmers get an idea of what the finished program is *supposed* to look like before they start programming. It might be a basic version of the program that handles only basic functions and the requirements might change during development but they still work out the requirements up front. This prevents misunderstandings and other problems that can crop up.

Requirements analysis is often done through conversations and meetings with the customer. The customer can be an outside company or people from other departments within the same company. This can take weeks or more, especially if the software needs to work with other systems or if the people involved realize that some of the requirements need to be changed in order for the software to be usable or even safe. Once the requirements are complete, they're often documented in different ways for use by the programmer and others in the process.

A Closer Look at Rogue

For this project, the requirements are actually demonstrated through two MS-DOS versions of Rogue – a public domain version released in 1984 and the commercial version released by Epyx, Inc. in 1985. The Epyx version is still available and playable through Steam for a small fee. Another online version was created by Donnie Russell II, an independent programmer, and is still available (http://donnierussellii.github.io/JSRogue.html). The versions are mostly the same except for minor differences in graphics and names. You can also view playthroughs on YouTube.

Before we start developing our own roguelike game, we still need to look at the existing game and put together an outline of its rules and gameplay in order to make one that's similar. This is sometimes called reverse engineering – analyzing a finished product in order to create a new version. Reverse engineering is sometimes explicitly prohibited in software user agreements but, since there is no user agreement for this public domain game and