Car Racing Video Game – Journey

# Purpose

The purpose of these notes is to document my journey in learning how to build a modern car racing video game. I am into the history of things, so I plan on exploring the evolution of car racing games from Grand Prix on the Atari 2600 to Outrun and modern car racing games. My main goal is to build examples of all these car racing games to improve on my knowledge of C++ and auxiliary tools (e.g. Tiled Map Editor) I will need to master to build games.

Unfortunately, I must explore modern C++. Nothing wrong with C++ just that I have not coded in C++ for over a decade (maybe more). I come from a strong C[[1]](#footnote-1) programming background but have been primarily a Java backend developer for the last twenty years. I see that C++ has changed quite a bit with the goal of adding more common data structures (e.g. <vector>) and modern programming concepts to make memory management safer (unique\_ptr).

The beginning of this journey will start less with car racing games and more into getting acquainted with C++, SDL2, and many other tools and applications. The more I learn the more I will start to build car racing games as we go through the techniques required to construct everything from Night Driver to Mario Kart to a more modern racing game[[2]](#footnote-2).

I will take two approaches to what I refer to as “Car Racing Video Game” (CRVG) project:

* ***Informal*** – Any tutorials, reading and code that is about learning how to make a game or use a tool
* ***Formal*** – Any writing and code that directly pertains to CRVG

# Phase 1 – Learning, Learning, Learning

Nobody in this industry knows what they’re doing, we just have a gut assumption based on the games that we can play. – Cliff Bleszinski

Cliff Bleszinski is famous for the ***Unreal Engine*** and ***Gears of War*** while at ***Epic Games***. Many games today are built from the Unreal Engine – ***Fortnight***, ***Black Myth: Wukong***, ***Borderlands 4***, and too many more to list[[3]](#footnote-3). The first step for me is to relearn C++ and some tools used to create video games like SDL2 and Tiled Map Editor.

The initial learning process will involve video tutorials, books and some online website tutorials. I will mention everything I find worthwhile in these notes.

## Following YouTube Tutorials

The first thing I decided to do was to follow some youtube tutorials that covered building video games with C++ or tools you would want to use in building games. I follow and take individual notes[[4]](#footnote-4) which hopefully provide more details. I look up a lot of information using Copilot and google searches since it is my experience that the video presenter does not cover every term and technique presented. In my notes, I may not provide attribution for all the code, definitions, and explanations but I will try.

## How to Make a 2D Game Engine with C++ & SDL2

This is a youtube playlist: <https://www.youtube.com/playlist?list=PL-K0viiuJ2RctP5nlJlqmHGeh66-GOZR_> that covers the development of a platformer type game using C++ and SDL2.

The video assumes you know C++ rather well and does not get into any language discussions – other than how to build classes in C++ that function as Singletons. The video also uses ***Code::Blocks*** are the IDE. I can understand its selection since it is available on all major operating system platforms. It could be the version of Code::Blocks but if felt like the presenter did not utilize all the cool features of Code::Blocks, for example, he seemed to prefer the Java style of braces but always just manually edited the generated files and functions rather than changing the source code style. I also noticed he always removed the destructor method the code automatically generated rather than just deselecting that option when a new class was created:

A white box with black text

AI-generated content may be incorrect.

Figure - Deselecting "Virtual destructor"

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Selecting the Source Formatter

I purchased the book written by the video presenter, Franc Poulhela, ***Cross-Platform 2D Game with SDL2***. The book was inspired by doing this video series but alas it is completely different. It uses Video Studio Code and is more in the spirit of writing a game engine and then using that game engine to create a Flappy Bird clone game. I decided to put off reading the book until I am done with this series. Like most series the first one had 87K views but by video #39 it drops to 10K and the last one in the playlist it is half that number. I think many people dropped off because the game assets after video #7 or so are not available unless you become a patreon member.

### What is a good style for C++ development?

I may be rusty in C++ development but the programming style used in this series feels off to me. I decided to use K&R formatting for my source. I wish more people would start a series discussing:

* Formatting preferences
* C++ code style preferences
* Testing infrastructure they will use (or not use)
* A review of the features of the IDE they select

I was not sure of the style of some classes that implemented everything in the header (\*.h) file.

|  |
| --- |
| Copilot question: Is it common the create C++ header files that does not use an implement file but implements everything in the header file?  Yes, it's fairly common in C++ to have **header-only implementations**, where all function definitions are written directly in the header file instead of a separate .cpp file. This approach is often used for:  **Common Use Cases**   * **Template Libraries**: Since templates must be instantiated at compile time, their definitions need to be available in the header file. * **Inline Functions**: Functions defined inside a class in a header file are implicitly **inline**, which can improve performance. * **Header-Only Libraries**: Some libraries (like Boost and Eigen) are designed to be **header-only**, simplifying distribution and avoiding linking issues.   **Pros & Cons**  ✅ **Advantages**:   * Easier to distribute (no need for compiled binaries). * Reduces linking complexity. * Improves performance for small functions via inlining.   ❌ **Disadvantages**:   * Can **increase compilation time** since every source file that includes the header must compile the function. * May lead to **larger binary sizes** due to redundant function definitions across translation units. |

## A Crash Course in C++

As part of the effort to learn modern C++ I am reading the book ***C++ Crash Course***, by Josh Lospinoso. The book is a ***No Starch Press*** (<https://nostarch.com/>) book. I purchased the book. Before I purchase a book, I try to check out the reviews on amazon.com.

A close up of text

AI-generated content may be incorrect.

Some publishers are known to produce high quality books and No Starch Press is one such publisher.

I admit I pick up a lot of books for free on pdfdrive.com but I like to pay for books I find useful it supports authors spending their time and energy providing high quality books. This book is perfect for programmers like me who are not familiar with modern C++, it is written of intermediate or advanced programmers in other programming languages.

The code and errata for the book can be found at <https://ccc.codes/>.

TBD: More comments…

## Pikuma Courses

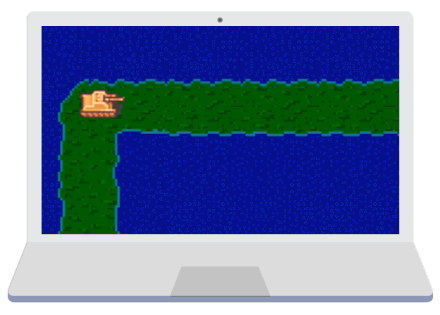
I found a website that offers at a reasonable price excellent courses to learn game programming - <https://pikuma.com/>. I am working my way through two courses:

* 2D Game Engine with C++ and Lua
* 2D Game Physics Engine Programming

The course lecturer Gustavo Pezzi explains everything clearly and completely. I never feel the need to go and look up some terminology or tool. You can always post queries for each lecture and know that you will get back a response in short order.

### 2D Game Engine with C++ and Lua

This is a perfect companion to the youtube video “How to Make a 2D Game Engine with C++ & SDL2”.



This course is a gentle and comprehensive introduction to the fundamentals of 2D game engine architecture. This course the same tools and libraries with the addition of GLM (math helper library), Dear UmGui (to create UI), Sol (a binding library for modern C++ and Lua), and of course Lua (a scripting language).

All the tools are cross-platform. The idea of building cross-platform applications has always intrigued me. I haven’t had to deal with the issues around cross-platform for 20 years because I have been a Java developer. I was able to compile on my Windows desktop my Java code into .class files that I then could upload to the target Linux machine and it would run with no issues.

I highly recommend the course.

Atari 2600

Grand Prix (top-down view)

Indy 500

Night Driver

Pole Position

Fatal Run

Enduro

1. I have a patent from a C program I built with working for AT&T as a graduate student (see: <https://patents.justia.com/inventor/lorraine-figueroa>) [↑](#footnote-ref-1)
2. I will never be able to create anything close to a modern racing game, but I can still demonstrate how to use a tool like Unity or Unreal Engine to build a better looking modern car racing game. [↑](#footnote-ref-2)
3. Games made with Unreal Engine - https://www.pcgamingwiki.com/wiki/List\_of\_games\_that\_use\_Unreal\_Engine\_5 [↑](#footnote-ref-3)
4. All my notes and projects are available on my nyguerrillagirl github site. I am currently using the repo 300-days-of-code, as I progress the repo name will change. [↑](#footnote-ref-4)