Daniel Adler

Professor Sepehr

COMP 141

10/02/2020

Which Game Engine Better Supports Game Scripting: Unreal Engine with C++ or Unity with C#?

1. Introduction

After the introduction of Machine code and low-level computing, programming languages have been developed for the purpose of making writing code more accommodating and efficient for those people who write and use it. Due to the wide variety of software that exists, programming languages have been built around a variety of paradigms used to accommodate different methods of computation. As a result, almost all game engines have adopted Object-Oriented programming due to the presence of Actors and Entities that operate as individual objects with their own attributes; Each requiring their own respective scripts and components. Two of the most popular game engines on the market seem to be Unity and Unreal Engine. While both provide similar interfaces, Unity appears to use C# while Unreal Engine sticks with C++. Both languages have proven to be reliable, as some of the most popular games, that do not utilize their own custom-made engines, have spawned from these. However, there are distinct differences in their scripting languages: While both are primarily object-oriented, C++ was designed with the intention of giving users the flexibility to more closely interact with hardware and allow for dynamic memory allocation and C# was created as a rival to Java with a major focus on Windows Desktop Application development. C# is intrinsically simpler to write in than C++, since memory allocation and garbage collection are handled, but C++ requires these things to be done manually by the user. C# is a higher-level language and therefore has more processing during compile time, while C++ is considered a high-level language that has less to handle. The differences go on, but both languages have specific attributes that allow them to excel in different areas. For this reason, both languages can operate comparatively to each other when used for scripting within each of these game engines.

2. The Purpose of Each Language

When evaluating how a language operates for a specific purpose, it is important to note the initial intended role of the programming language within computing.

2.1. Origins

C# was originally created to be an improvement of Java after the failure of IE3, an enhancement of the JVM that Microsoft was sued for due to its intense similarity to the Java 1.1 standard (Forbes, 2018). Later, C# was developed using its own .NET framework independent of JVM and was able to bring with it many new advantages. C# offers access to both value and reference semantics with its class and struct types which can improve performance, offers deeper integration with Windows technologies, and can support multiple inheritances. In this way C# was able to become competitive with Java for various purposes. On the other hand, C++ was created to breach the divide between the low level programming offered by C and the object oriented programming offered by SIMULA in order to achieve high level abstraction that was close enough to hardware to operate efficiently (Stroustrup, 2010). It was due to these purposes that C# and C++ were able to thrive: C++ offers the possibility of object oriented programming that can run in times comparable to C, and C# offers a framework that can support application development for video games especially.

2.2. Drawbacks

However, each of these languages also come with specific disadvantages in their intrinsic design. Due to C++'s low-level components, there is more precision required in programming to prevent memory leaks and prevent crashing, threads are a new concept within C++ that are only currently supported via Lambda functions, and security issues exist with the presence of global variables, pointers, and friend functions (Data-Flair, 2020). C# is not a perfect language either, with lacking access to hardware or the face of requirement of the .NET framework limiting portability forcing servers for C# applications to be run on strictly Windows machines. Both languages have built in components that make each of them adequate for the topic of game design. With C#, since most complex games run on Windows devices, the .NET framework is often complementary to the design of these types of games.

3. Simplicity

In addition to their varying purposes, C++ and C# also have differences in their simplicity in their IDEs, syntax, and semantics that make their features accessible to the developers that user them.

3.1. IDEs

While they both offer options to script using Visual Studio Code which is renowned for its accessible interface, they each offer other unique IDEs such as Eclipse, Dev C++, and Code::Blocks for C++ as well as JetBrains Rider, MonoDevelop, and Browxy for C#.

3.2. Syntax

In addition to IDEs, both languages have characteristics in their syntax that may make writing certain scripts and programs simpler or more complex.

3.2.1. Similarities

C++ and C# have many similarities on the surface that make them comparable languages. C# has all the data types that C++ has, their if statements function the exact same way, brackets are required to mark the presence of scopes, and semicolons are required between statements to signify a separation. However, although these languages have attributes that make them look similar on the surface, they still have far more syntax differences than similarities.

3.2.2. Classes and Structs

Other differences can be found in the way that each language handles data: Classes and Structs for instance are handled in different ways: Regarding Inheritance, C# has public inheritance that does not require an access modifier on the base class' name, classes can only inherit from one base class which makes object oriented programming less powerful, all objects inherit from the System.Object class.

Aside from inheritance, C# includes two more access modifiers: Internal and Protected Internal, which allow access to a that variable only by classes that are derived from the class that the variable exists in (Microsoft, 2015). While C++ has variables, functions, constructors, deconstructors, and operator overloads as class members, C# also offers delegates which are unique function types that allow you to pass other functions in as parameters similar to pointers in C++, events which are functions that are called as a result of events occurring, and properties which are prebuilt ways of accessing data members similar to getter and setter functions. Both languages are able to have destructors, but they work differently in each one: Since dynamically allocated memory in C# is deallocated automatically, the use of deconstructors is rare, but necessary if there is a specific reason to eliminate dynamically allocated memory before the program stops running. To accomplish this in C#, the class using the destructor must derive from the IDisposable interface and implement the Dispose function. This enables access to the .Finalize function which is similar to using the '~' character before the constructor name to initialize a destructor in C++. C# also varies from C++ in the way it handles Structs. In C++, we are used to structs being handled very similarly to classes, but default access and default inheritance are public. In contrast, C# utilizes structs similarly to how they are used in C, where they cannot have any default constructors and lack functions or inheritance.

3.2.3. Error Handling

As far as Error Handling goes, C++ and C# function the same on this topic except for a few distinctions. C# requires all instances of classes that throw exceptions to derive from the System.Exception but introduces the finally block that executes as soon as a try or catch block is exited from.

3.3. Semantics

C++ and C# have many similarities in their typing that make both languages similar when reading through and during compile time that give each language different functionalities during different circumstances.

3.3.1. Typing

C++ and C# both use explicit typing although C# has the capacity for implicit typing with the use of the var keyword. C# uses strictly static typing just like C++ and is strongly typed, where C++ is considered for the most part strongly typed but does provide some capability to undermine the typing apparatus (Stack Overflow, 2014). Some syntactic differences stem from C#'s strong typing and C++'s weak typing: While allowed in C++, C# is not able to handle instances of numeric variables in Boolean expressions unless they are defined as such and Wrapper classes must be type-casted before being utilized.

3.3.2. Scoping

Both languages use strictly static scoping due to variables declared only existing within the scope that they have been assigned to.

3.3.3. Reference vs. Value

With the integration of pointers, C++ has ease of access to both reference and value semantics due to the presence of pointer manipulation. C# also has access to reference semantics, but this is primarily provided by some of its default types such as string, and there is not the same level of flexibility that C++ offers.

4. Performance

Within Game design, performance is one of the most important criteria for evaluating the quality of the game. Games with high FPS drops, low server quality, and limited support for detailed graphics often become obsolete and looked down upon when compared to games that do not possess these flaws. Low memory efficiency can be cause for concern and limit the available audience for the game you create due to the need for satisfactory hardware. For this reason, it is essential that the scripting languages behind these games can efficiently handle the data it uses.

4.1. Memory Management

One of the most notable differences between C# and C++ is that C#, like Java, runs on a virtual machine instead of on hardware, meaning that memory allocation and deallocation is handled automatically. While this may be an advantage due to the simplicity of not having to manage memory, this can potentially pose larger performance issues should the size of the game you are creating grow to be too large. C++ offers flexibility in its programming comparable to C, which allows for dynamic memory allocation to be completely managed by the user but also poses the risk of generating garbage in the hands of an inexperienced programmer. However, this type of memory management is more important for games that are being developed for console, as those typically lack some form of backup that can make poor memory allocation much more dangerous. Hence, if a game is more performance intensive, C++ is a safer pick.

4.2. Compilation Time

While C# is an interpreted language whose code is converted to byte code which is handled by the virtual machine after compile time whereas C++ is compiled down to machine code. While this means that C++ can often run approximately between 5 and 10 times faster than C#, having an interpreter allows C# to operate on any device so long as it has the .NET framework. So, depending on what kind of software that needs to develop, both programming languages are viable options. But C++ will always be more applicable when performance is in question.

5. Conclusion

All in all, C# and C++ are both promising languages for game development, each with their own distinct advantages. C++ has the capability for more dynamic memory allocation and compiled code allow it to run much faster and efficiently than C#. However, C# has more support for inexperienced developers and Windows applications, as well as support for unique object-oriented programming function types. Overall, C++ proves to be a more effective language if the game you are designing is memory intensive, requires a higher FPS for optimal gameplay, or depends heavily on servers for its operation. However, C# is not useless; Its unique syntax and semantics may be applicable within the development of specific types of games, successful games such as Hearthstone, Kerbal Space Program, Terraria, Magicka, and Temple Run have run on C#, and the Unity interface offers an interface independent of what Unreal Engine does that might be preferable to certain users. In the end, the game engine best for you is still a decision based off personal preference and the goals of your software.

References

Sweeney, T., 2020. *Unreal Engine 4 Documentation*. [online] Docs.unrealengine.com. Available at: <https://docs.unrealengine.com/en-US/index.html> [Accessed 3 October 2020].

EDUCBA. 2020. *C++ Vs C# | Know The Top 7 Most Important Differences*. [online] Available at: <https://www.educba.com/c-plus-plus-vs-c-sharp/> [Accessed 3 October 2020].

Guru99.com. 2020. *C++ Vs. C# - What'S The Difference?*. [online] Available at: <https://www.guru99.com/cpp-vs-c-sharp.html> [Accessed 3 October 2020].

Devdocs.io. 2020. *Devdocs — C++ Documentation*. [online] Available at: <https://devdocs.io/cpp/> [Accessed 3 October 2020].

Docs.microsoft.com. 2020. *C# Docs - Get Started, Tutorials, Reference.*. [online] Available at: <https://docs.microsoft.com/en-us/dotnet/csharp/> [Accessed 3 October 2020].

Forbes. 2020. *Why Did Microsoft Create C#?*. [online] Available at: <https://www.forbes.com/sites/quora/2018/03/02/why-did-microsoft-create-c/?sh=25aa195270f3> [Accessed 16 November 2020].

Big Think. 2020. *Why I Created C++*. [online] Available at: <https://bigthink.com/videos/why-i-created-c> [Accessed 16 November 2020].

DataFlair. 2020. *Advantages And Disadvantages Of C++ | Make Your Next Move! - Dataflair*. [online] Available at: <https://data-flair.training/blogs/advantages-and-disadvantages-of-cpp/> [Accessed 16 November 2020].

C-sharpcorner.com. 2020. *Difference Between C++ And C#*. [online] Available at: <https://www.c-sharpcorner.com/uploadfile/gtomar/difference-between-cpp-and-C-Sharp/> [Accessed 16 November 2020].

interpreter?, I., Andreson, M. and Dimitrov, D., 2020. *Is .NET VM A Compiler Or An Interpreter?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/3275360/is-net-vm-a-compiler-or-an-interpreter> [Accessed 16 November 2020].