Using Game Development to Teach C#

Lacey Weeks
Computer Science
University of North Alabama
Florence, Alabama, United States
lweeks1@una.edu

ABSTRACT

In this paper, I focus on using game development (or gamification) as a teaching method to teach C# programming. Game development is an attractive way to teach different programming languages and computer science concepts. There are a variety of studies that demonstrate the interest in teaching via gamification and that this method has benefits such as motivating and encouraging students to be interested in their classwork, help these students play an active role in their learning, and is an overall fun way to learn something new and challenging. I will discuss some of those studies within this paper to demonstrate the importance of this technique. I will also cover the historical background and a language overview of the C# programming language such as who created the language and why, the impacts of C#, and some finer details of the language such as its capabilities, syntax, semantics, and design implementation alongside some examples. Afterwards I will discuss my approach in solving my research question which will include which game engines I looked at and the one I had chosen. For results, there are a few articles that support using gamification (or game development) whose results I will discuss further in detail later alongside my experience with using the Unity game engine and their tutorials. Finally I will end this paper with any future work I would like to do with C# such as extension to this topic and any other questions I have regarding C#, such as testing my research idea out on multiple programmers (both beginners and experiences), comparing and contrasting the two groups of programmers and how they learn via this method, and even potentially seeing if there are certain game engines that are better for the learning environment than others.

KEYWORDS

C#, programming, game development, game engine, game, teaching, gamification, tutorials, Unity

1 Introduction

C# can be applied to many fields, from geographic information systems (GIS) to statistics to web development. I had chosen C# originally for its connection to GIS and statistics. However, as I learned more about this language and its applications I wandered into game development as it does appeal to the artistic/storytelling side of my personality. Game development appeals to me because it allows me to design and build the stories that I imagine when before I wasn't quite sure on how to express those ideas.

This research paper will focus on using game development as a method to teach C# programming. The goal is to see if this method can successfully teach C# to any individual who is new to C# whether they are new to programming as a whole or a more experienced programmer that decided to add C# to their growing collection of programming languages.

Game development has the potential to help teach C# up to a degree if used correctly. It can help users get started on learning the language while also giving them opportunities to have fun while learning. However, there can be a risk of this approach teaching bad coding habits or not giving an in-depth understanding of the C# language depending on how this teaching method is used. For example, if a user is new to programming and is trying to self-teach using this method they might be more susceptible to picking up on bad coding habits compared to learning by gamification in a classroom taught by a computer science professor. If anything else, game development should be used to encourage and motivate users to learn C# programming.

2 Background

This section will cover the historical background of C#, an overview of the language, and the literature review regarding my research topic.

2.1 Historical Background

C# is an object-oriented programming language created by Microsoft and released in 2002. Since it is a Microsoft made language, it is designed to run on the .NET Framework. It is similar to Java, C++, and C. Since its creation, there has been updated versions of C#, the latest being C# 8 which was released back in September 2019. With each new version, new features are added to improve the language [1].

The next two figures, Figure 1A and Figure 1B, will show a list of the current C# versions and the important features that had been introduced in each version [2] [3] [4].

Version	.NET Framework	Visual Studio	Important Features
C# 1.0	.NET Framework 1.0/1.1	Visual Studio .NET 2002	- Basic Features
C# 2.0	.NET Framework 2.0	Visual Studio 2005	- Generics - Partial Types - Anonymous Methods - Iterators - Nullable Types - Private Setters (Properties) - Method Group Conversions (Delegates) - Covariance and Contra-Variance - Static Classes
C# 3.0	.NET Framework 3.0/3.5	Visual Studio 2008	- Implicitly typed local variables - Object and collection initializers - Auto-Implement properties - Anonymous types - Extension methods - Query expressions - Lambda expressions - Expression trees - Partial Methods
C# 4.0	.NET Framework 4.0	Visual Studio 2010	- Dynamic Binding (late binding) - Named and optional arguments - Generic co- and contravariance - Embedded interop types - Async features - Caller information

Figure 1A. Shows the first four versions of C#.

Version	.NET Framework	Visual Studio	Important Features
C# 5.0	.NET Framework 4.5	Visual Studio 2012/2013	- Async features - Caller information
C# 6.0	.NET Framework 4.6	Visual Studio 2013/2015	- Expression Bodied Methods - Auto-property initializer - name of Expression - Primary constructor - Await in catch block - Exception Filter - String Interpolation
C# 7.0	.NET Core 2.0	Visual Studio 2017	- Out variables - Tuples - Discards - Pattern Matching - ref locals and returns - local functions - More expression-bodied members - throw Expressions - Generalized async return types - Numeric literal syntax improvements
C# 8.0	.NET Core 3.0	Visual Studio 2019	- Readonly members - Default interface methods - Using declarations - static local functions - Disposable ref structs - Nullable reference types - Asynchronous streams - Asynchronous disposable - Indices and ranges - Null-coalescing assignment - Unmanaged constructed types - Stackalloc in nested expressions - Enhancement of interpolated verbatim things

Figure 1B. Shows the last four versions of C#.

C# can be used in many areas such as mobile applications, desktop applications, web applications, web services, websites, games, VR, and database applications, just to name a few. As we can see, C# is used in a variety of areas, but why do people love using this language? Well, it could be for a variety of reasons such as it is a popular language, easy to learn, simple, has a huge community support, it is object-oriented, and is close to C, C++, and Java which makes it easier for programmers coming from those languages to dive into C# [1].

2.2 Language Overview

C# is considered an object-oriented programming language. It is also considered a type-safe language. The syntax is considered to be both highly expressive and easy to learn. If you are familiar with C, C++, or Java then you should have no difficulty learning C#. According to the C# documentation page on Microsoft website, C# syntax is designed to simplify "many of the complexities of C++" while providing "powerful features such as nullable types, enumerations, delegates, lambda expressions, and direct memory access" [5]. It also supports encapsulation, inheritance, and polymorphism. Next I'll discuss the syntax, semantics, and design implementations in further detail.

To be able to code in C#, there are a few naming guidelines that the programmer will have to follow. For this paper, I will only list about a couple of these naming conventions. Capitalization conventions and general naming conventions. With the general naming convention, when naming a variable or word, choose an easily readable name. HorizontalAlignment is more readable AlignmentHorizontal. Aim for readability over brevity. CanScrollHorizontally is more understandable than ScrollableX. You want to choose names that are easily readable but also explains what this variable is supposed to do. Avoid using underscores, hyphens, and just any nonalphanumeric characters. It also best not to use Hungarian notation when naming variables either. Also try to avoid any names that "conflict with keywords of widely used programming languages" [6]. Avoid abbreviations and acronyms when naming your C# variable. Try to use more interesting names such GetLength over GetInt. Overall, just try to use a specific, straight-forward name that helps describe what the variable is for without making it too long or complicated.

Next is the capitalization convention. It is suggested to use capitalizations to differentiate words in a variable's name rather than underscores. There are a couple of ways that are acceptable in C#. PascalCasing and camelCasing. PascalCasing "capitalizes the first character of each word" [7]. An example would be ComputerScience. The camelCasing also "capitalizes the first character of each word" with the exception of the first word [7]. An example would be cyberSecurity.

There are a few different scopes in C#. "The scope of a variable determines its visibility to the rest of a program" [8] which means depending on where the variable is located in the code will determine how visible it is to other parts of the code. Class-level scopes, method-level scopes, and nested scopes can determine the scope of a variable. In class-level scopes, a variable that is defined in a class can be available "to any non-static method within the class" [8]. With the method-level scopes, variable's scope is restricted to the method that it is used in. Anything in that method can use that variable, but no blocks of code can use the method's variable outside of the method. The nested scope is similar to the

method scope since it restricts the variable's scope to a specific location. In this nested scope's case, if a if statement or for loop uses a variable and this variable has only been initialize in that if statement or for loop, then the variable cannot be used anywhere else outside of those if statements and for loops, not even in the method itself. Here is an example of a method-level scope and a nested scope using C# syntax. You will notice that in the nested scope the message variable is restricted to the if-else statement and cannot work outside of the if-else statement.

```
static void Main(string[] args)
{
    int score;
    score = 100;

    if (score >= 50)
        Console.WriteLine("Good score : {0}", score);
    else
        Console.WriteLine("Poor score : {0}", score);
}
```

Figure 2A. Method-level scope

```
static void Main(string[] args)
{
    int score = 100;

    if (score >= 50)
        string message = "Good score";
    else
        string message = "Poor score";

    Console.WriteLine(message);
}
```

Figure 2B. Nested scope

C# offers around 13 value types. They are bool (Boolean), byte (8-bit unsigned integer), char (character), decimal, double, float, int (integer), long (long integer), sbyte (8-bit signed integer), short (short integer), uint (unsigned integer), ulong (unsigned long integer), and ushort (unsigned short integer) [9]. Here are a few examples of what they would look like in C# code if you were to implement them.

```
byte x;
int x = 1;
double y = 3.14;
float z = 3.15F;
bool answer = true;
char t = "t";
string name = "Computer Science";
```

Figure 3. Type examples

Expressions are "a string of operators and operands." [10] There are several constructs that can act as operands, such as literals, constants, variables, method calls, and element accessors. The operators can take one, two, or three operands. Then the expressions can be combined with these operators in order to create more expressions. Here's an example of one of the constructs, a literal construct. There are a few different literals such as integer, double, float, boolean, character, and string as you can see in Figure 4 [10].

sample.cs

```
1  static void Main()
2  {
3
4      Console.WriteLine("{0}", 1024);          //  inte literal
5      Console.WriteLine("{0}", 3.1416);        //  double literal
6      Console.WriteLine("{0}", 3.1416F);       //  float literal
7      Console.WriteLine("{0}", true);          //  boolean literal
8      Console.WriteLine("{0}", 'x');           //  character literal
9      Console.WriteLine("{0}", "Doctor Who");  //  string literal
10
11 }
```

Figure 4. Literals

Due to C# having "more than 45 operators and 14 levels of precedence" [10] evaluating expressions can be a little bit complex. Depending on the order you evaluate an expression, one expression can have multiple answers. An example would be: 3 * 5 + 2. If you do multiplication first then the answer would be 17 while addition first would result in the answer being 21. C# has a couple of ways of dealing with this, one being precedence and the other being associativity. Precedence deals more with the operators with the highest level is completed first then the second highest, and so on until the expression is solved. Associativity focuses more on expressions where the operators that have the same level of precedence. Here's an example of how an expression will look in C#.

```
int add = 5 + 6;
int substract = 12 - 3;
int multi = 3 * 4;
int q = 10 / 3;

double x = 7 - 5 * 2 + 4;
```

Figure 5. Expressions

C# has a few different control structures such as if statements, if-else statements, switch statements, for loops, while loops, do-while loops, and foreach loops. If statements are intended to initialize a block of code if a certain condition is met, else that block of code will not be executed. If-else statements build upon this by including an else condition where if the first if condition is not met then to check a second condition (the else condition) and execute the block of code associated with the else condition if the criteria has been met. For example, if you go to class and there is a test then you take the test, else you take notes and prepare for an upcoming test. Switch statements are very similar to the if and if-else statements since it depends on conditions and how the user responds. Think of a calculator. Depending on which button you press, it will either add, subtract, multiply, or divide the numbers you provided. A switch statement will have a form of if statements for each operator. If the user types '+' then it would add the numbers provided, else if the operator provided is a '-' then it will subtract the numbers, and so on. Figure 6A and Figure 6B will show examples of if, if-else, and switch statements [9].

```
int sample = 5;

if(sample > 0) {                                    // If-else statement
    Console.WriteLine("Positive");
} else {
    Console.WriteLine("Negative");
}

if(sample == 5) {                                   // Basic if statement
    Console.WriteLine("The answer is 5");
}
```

Figure 6A. if and if-else statements

```
int a = 5;
int b = 7;
int sum;
int dif;
int prod;
int quot;
char thing = "+";

switch(thing) {             // Takes in an input called thing
    case "+":               // Depending on what thing is, it will
        sum = a + b;        // either trigger the addition case, the
        break;              // subtraction case, the multiplication case,
    case "-":               // or the division case
        dif = a - b;
        break;
    case "*":
        prod = a * b;
        break;
    case "/":
        quot = a / b;
        break;
}
```

Figure 6B. A general switch statement for a calculator.

While loops, do-while loops, for loops, and foreach loops build upon the if, if-else, and switch statements. They are like the if statements since they require a condition to enter the loop and to only execute certain code if those conditions are met. The difference is that the programmer will need to include an ending of some sort in the condition else you will have a loop that goes on forever. An example would be putting gas in your vehicle. While the gas tank is not full, keep pouring gas into the tank. The while loop and do-while loop are almost identical with one exception. With while loops, they start off with a while(condition) then { a block of code }; and do-while loops while have a 'do' at the beginning then { a block of code} and finally ending with the while(condition). Figure 7A and Figure 7B will show an example of these two loops [9].

```
while(num > 0) {
  mag++;
  num = num / 10;
};
```

Figure 7A. Shows the normal while loop

```
do {
  nextdigit = num % 10;
  Console.Write(nextdigit);
  num = num / 10;
} while(num > 0);
```

Figure 7B. Shows the do-while loop

The for loop and foreach loop builds upon the while loop concept. In the for loop you need to include an initialization, condition, and iteration. Of course, whatever variable you initialize in the for loop will only be associate with that for loop and you will not be able to call that variable outside the for loop. Figure 7C will show an example of what a for loop would look like in C#.

```
// For loop
for (int x = 100; x > -100; x -= 5) {   // X starts at 100 and decrements by 5
    Console.WriteLine(x);               // while x is greater than -100
}
```

Figure 7C. For loops

C# also have a few abstract data types (ADTs) such as stacks and queues. Stacks follow a last-in, first-out (LIFO) approach which means that the last item put on the stack is the first item to come out. Stacks have three operations which are push, pop, and peek. Push will add an item on the top of stack, and pop will remove the top item from the same stack. Peek is for looking at the top item without removing the item [11]. An example of a stack would be a pile of books. You put a book down then put a second book on top of the first, then a third book on top of the second, and so on. In this case, the first book into the book stack is on the bottom of the stack and will be difficult to grab without taking the books on top of it off first. In order to reach it you would have to take the top book on the stack off, in this case the last item in the stack must be taken off

first. Here's an example of what a stack would like in C#. The code example comes from Microsoft's website [12].

```
using System;
using System.Collections;
public class SamplesStack  {

   public static void Main()  {

      // Creates and initializes a new Stack.
      Stack myStack = new Stack();
      myStack.Push("Hello");
      myStack.Push("World");
      myStack.Push("!");

      // Displays the properties and values of the Stack.
      Console.WriteLine( "myStack" );
      Console.WriteLine( "\tCount:    {0}", myStack.Count );
      Console.Write( "\tValues:" );
      PrintValues( myStack );
   }

   public static void PrintValues( IEnumerable myCollection )  {
      foreach ( Object obj in myCollection )
         Console.Write( "    {0}", obj );
      Console.WriteLine();
   }
}


/*
This code produces the following output.

myStack
    Count:    3
    Values:    !    World    Hello
*/
```

Figure 8A. A stack in C#

With queues, on the other hand, the item first in is the first item out. In other words, it follows a first-in, first-out (FIFO) approach. A queue only has two operations, an enqueue and a dequeue. Enqueue, like a push in stack, adds an item to the top of a pile. The dequeue removes an item from the bottom of the pile which is different from a pop. A stack's pop would remove top items while a queue's dequeue removes bottom items [11]. A good example would be a checkout line at a grocery store. The first customer in the checkout line is the first person to be able to pay for their groceries and leave the line. Here's an example of a queue in C#. The example is from Microsoft's website [13].

```
using System;
using System.Collections;
public class SamplesQueue  {

   public static void Main()  {

      // Creates and initializes a new Queue.
      Queue myQ = new Queue();
      myQ.Enqueue("Hello");
      myQ.Enqueue("World");
      myQ.Enqueue("!");

      // Displays the properties and values of the Queue.
      Console.WriteLine( "myQ" );
      Console.WriteLine( "\tCount:    {0}", myQ.Count );
      Console.Write( "\tValues:" );
      PrintValues( myQ );
   }

   public static void PrintValues( IEnumerable myCollection )  {
      foreach ( Object obj in myCollection )
         Console.Write( "    {0}", obj );
      Console.WriteLine();
   }
}
/*
This code produces the following output.

myQ
    Count:    3
    Values:    Hello    World    !
*/
```

Figure 8B. A queue in C#

2.3 Literature Review

There are a few books and several tutorials that can be found online that teaches users C# programming via building games. "Learning C# by Programming Games" by Egges, Fokker, and Overmars and "Unity 5: Learning C# by Developing Games" by Lukosek, Doran, and Dickinson are a couple of book examples. Game engines such as Unity and Xenko have tutorials posted on their websites designed to help teach users C# programming via building games using their game engines. Since there are books and tutorials regarding learning C# via game development (building games) it is reasonable to assume that this method to teach C# programming can be used.

As for actual research regarding using game development as a teaching method rather than just tutorials, there are several papers related to the topic. Most of the papers I have encountered so far regarding using game development as a teaching method focused on topics such as using gamification to teach Java, improve students' motivation and engagement in the classroom, and even to "support active learning, experiential learning, and problem-based learning, thus encouraging students to play an active role in the learning process" [14]. Next I will briefly discuss some of those articles and their ideas regarding using gamification as a teaching method.

The first paper is "Gamification for Teaching Java." Even though this is not specifically for the C# programming language, it does relate to my research interest and Java so happens to be similar to C#. This paper first discussed a little bit about the background of gamification in the classroom. One of the studies provided in the background suggested that there was an "increase in attendance and

in course completion rates" [15]. However, there were also issues in studying the benefits of gamification such as "lack of a standard definition for the term, theoretical foundations are inconsistently referenced and interpreted, there is a gap between theory and practice of the platforms, and more empirical studies are need to understand the actual effects of gamification" [15]. There does appear to be some issues of what gamification is and how it should be study. The authors then went on to discuss setting up a Java course at the University College Dublin to test out the gamification approach on a group of students. They discuss the process of how they setup the class and how they designed the gamification class. Afterwards they did an evaluation using a short questionnaire to determine students' thoughts regarding the gamification approach.

Next is "Improving Participation and Learning with Gamification" which covers applying gamification to the classroom and using it to help improve student engagement. The authors experimented with the gamification concept by gamifying an MSc course. Student behaviors from both the gamified version and the non-gamified version of the class, which was from the previous year, were compared. In the first year of gamifying the class, students were more motivated and interested in their gamified class than they were in their non-gamified courses. Their "lecture attendance, attention to reference material and both participation and proactivity on the course's online forums greatly improved" [16]. However, their final grades did not change much despite being more motivated and engaged in their gamified course. The authors decided to do a new study on gamification based off the first year experience to see if more complete data can support their previous findings, understand how "the gamified experiment affect the grades" [16], and to see how the "student engagement is affected by the second gamified edition of the course" [16].

Finally, there is "The Designed Gamification Application Architecture and Elements for a C# Programming Course." This paper focuses more on designing and implementing gamification architecture and elements for a C# programming class. The author also notes that games can support learning and encourage students to take more of an initiative in their learning. However, they noted that gamification can fail if not done properly. For example, if the gamification-based solutions are "created on impulse" [14] or mixed "bits and pieces from game components without a clear and formal design process" [14] then they are more likely to fail.

3 Methodology

First, I will search for a few different game engines that will allow the user to program in C# as they build games. A few game engines that caught my interest as potential game engines to use in this research are Unity, Xenko, Godot, and Cryengine.

Second, I will check to see if my chosen game engines offer tutorials on building games using C# programming. The listed game engines in the previous step offer tutorials on not only how to build games in their game engines, but also tutorials teaching C# programming via building games in their game engines.

Third, I will experiment with at least one of my chosen game engines and their 'learning C# by building games' tutorials to check if I, as an amateur programmer who is new to C#, can learn how to code in C# via building games. For this research paper, due to limited time I have for this paper, I will focus mainly on Unity as my chosen game engine. Unfortunately, due to limited time I will not be able to experiment with other programmers, both beginners and experienced. Any tested results would be from my own experience testing my research idea. Fortunately, there is research that other people conducted regarding using gamification as a teaching method that will be used to compliment my conclusion and research.

4 Results/Conclusions

I went through a few videos under the Beginner Scripting tutorial on Unity's website in order to see if I could learn C# through their method. Note, Beginner Scripting is labeled as a project on the Unity's website, and so far from the different parts I looked at, each part consists of a short video and sample code. Going through the first three parts under this tutorial I felt like I did not learn that much regarding C# due to the fact that some of the programming concepts they introduced I was already familiar with due to the programming courses I had taken.

The first part or section is called "Scripts as Behavior Components." This particular part had a short video of roughly three minutes long discussing the sample code provided. The presenter discussed what the code is doing without directly mentioning the code itself. For example, the presenter pointed out in his video that you can change the color of the object (in this case, a sphere) by pressing a key ('r' for red 'g' for green, or 'b' for blue). However, there was no mention of code nor did he discussed how he made it where the sphere can change color at the press of the key. The presenter demonstrated the sample code in this case, but he never explained how the code is related to the event.

The second part, which is labelled "Variables and Functions," did a better job at explaining some of the concepts. The presenter briefly discussed what a variable and a function are while also recording his screen while he worked on the sample code. It felt like a walkthrough as he explained each step. However, I have a similar reaction to this part as I did to the previous part. As someone who has experience with programming, I already understand the concepts he discussed and the only benefit I had was seeing what the C# syntax looked like for variables and functions. If you had a beginner programmer, on the other hand, then I can understand having a video discussing these concepts as you walk the user through the code. This can be somewhat useful for the beginner programmers, but maybe not as much for the more experienced programmers.

The third part, "Conventions and Syntax," is more comparable to the second part than the first since it focuses more on explain concepts as the presenter walks through the code rather than merely demonstrate what the code can do without diving into the code and explaining it. In this section, he did cover a little bit about the syntax, mentioned the dot operator, compound elements, semicolons, indentation, and comments. I do like the explanation he gave for the dot operator where the variable on the left side of

the dot can be looked as a country and the variable on the right of the dot can be seen as a city within that country. For example, here's a snippet of code he included in the sample code.

```
Debug.Log(transform.position.x);
```

Debug can be seen as a country while Log can be seen as a city that resides in Debug. In other words, Log is a part of Debug. The same thing can be said for the transform.position.x except in this case x is like a street that can be found in the position city which resides in the transform country. I do like that he made this sort of comparison because it help gave me a better understand of the dot operator. Continuing on with comments and semicolons, he does explain how you can make a single line comment using // and for the comment blocks you begin with a /* and end with */. He also stated that any code within the comment block will not get executed. As for the semicolons, he stated that semicolons terminate a line, but you don't need them after ever closing curly brackets.

Overall, I think the Beginner Scripting tutorial is decent, but could be a lot better for teaching. Experienced programmers will not learn much from this particular tutorial while beginner programmers might find some useful information regarding C#. It would have been a better learning experience if there were more to this project than a short video with sample code. For example, I would have like to have seen some coding exercises for the user to try after watching each video. I do like that the presenter tried to connect it to games in some way such as having experience and level variables and properties and functions to calculate the experience and level of a player. There are better tutorials that not only teach C# but also blend the game development aspect with it than this particular tutorial [17].

According to a few of the papers mentioned in the literature review, gamification can help students. For example, in the "Improving Participation and Learning with Gamification" paper, the authors discussed their student feedback near the end. There was a questionnaire given to the students at the end of the semester and the students responded that the gamified course was more motivating and interesting than their other courses. However, there were more to work to the class, though not any more difficult. Of course, the "students mildly felt that they were playing a game instead of just attending a regular course" [14].

In conclusion, I do think game development (or gamification) can be a useful tool in teaching C# to students. However, it depends on how well the class is designed. If it is done on impulse without any clear direction or design then it is doomed to fail. Despite the issues that gamification has encountered it deserves a chance. If anything, it could be used to motivate students and encourage them to learn more about the programming language. It is an interesting and fun way to learn a new programming language.

5 Future Work

For future work, one thing I would focus on is comparing beginner programmers who are learning their first or second programming language with a group of more experienced programmers who have a few years or more experience with programming and most likely have experience with two or more programming languages by this point in their careers. The idea here is to see how the two groups compare and differ in learning a new programming language via game development. It would also be interesting to see which gamification approach work best for each group. For example, the more experienced programmers might be able to skim over certain information while the beginner programmers might need more explanations and details covering the topics that the experienced programmers skimmed over. How do these two groups differ in their understanding regarding using game development to teach C#?

Another thing I would like to focus on for a potential future research paper would be comparing different game engines and their tutorials for learning C# via building games. Each game engine's interface is different, and they all have their own unique learning curve. So, which game engine would be better for students to learn C# on? Do certain game engines offer better tutorials regarding learning C# than other game engines? If so, which game engine would be the best one for a student who is interested in learning C# via game development to learn on? Would more experienced programmers who are learning C# prefer one game engine over another? How does this compare to the game engines that beginners prefer?

Another research idea I would like to try if I were to continue down the game route with C# is to build a game that teaches C# programming. Instead of learning by building a game, a user can learn from playing a game.

There are a few other questions I have regarding C#: (a) What are the security vulnerabilities in C#, why do these vulnerabilities exist, and is there a way of dealing with these vulnerabilities in C#? (b) Why should a user use C# in image processing? (c) Can C# be used in machine learning? If so, what can C# do for machine learning? (d) What impacts does C# have on the Geographic Information Systems (GIS) field?

REFERENCES

[1] "C# Introduction," W3Schools, [Online]. Available: https://www.w3schools.com/cs/cs_intro.asp. [Accessed 10 April 2020].

[2] "C# Version History," [Online]. Available: https://www.tutorialsteacher.com/csharp/csharp-version-history. [Accessed 10 April 2020].

[3] "What's New in C# 7.0," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7. [Accessed 10 April 2020].

[4] "What's New in C# 8.0," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8. [Accessed 10 April 2020].

[5] "Introduction to the C# language and the .NET Framework," Microsoft, 20 July 2015. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/getting-

started/introduction-to-the-csharp-language-and-the-net-framework. [Accessed 10 April 2020].

[6] "General Naming Conventions," Microsoft, 22 October 2008. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/general-naming-conventions. [Accessed 11 April 2020].

[7] "Capitalization Conventions," Microsoft, 10 October 2008. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/capitalization-conventions. [Accessed 11 April 2020].

[8] R. Carr, "C# Variable Scopes," 11 August 2007. [Online]. Available: http://www.blackwasp.co.uk/CSharpVariableScopes.aspx. [Accessed 11 April 2020].

[9] H. Schildt, C# 4.0: The Complete Reference, McGraw-Hill/Osborne, 2010.

[10] D. M. Solis, Illustrated C# 2010, Apress, 2010.

[11] R. Miller, C# Collections: A Detailed Presentation, Pulp Free Press, 2012.

[12] "Stack Class," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.collections.stack?view=netframework-4.8. [Accessed 11. April 2020].

[13] "Queue Class," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.collections.queue?view=netframework-4.8. [Accessed 11 April 2020].

[14] L. M. Padirayon, "The Designed Gamification Application Architecture and Elements for a C# Programming Course," in *ICMSSP 2019: Proceedings of the 2019 4th International Conference on Multimedia Systems and Signal Processing*, New York, NY, USA, 2019.

[15] Ç. Çubukçu, B. Wang, L. Goodman and E. Mangina, "Gamification for Teaching Java," in *SIMUTOOLS '17: Proceedings of the 10th EAI International Conference on Simulation Tools and Techniques*, New York, NY, USA, 2017.

[16] G. Barata, S. Gama, J. Jorge and D. Gonçalves, "Improving Participation and Lering with Gamification," in *Gamification '13: Proceedings of the First International Conference on Gameful Design, Research, and Applications*, New York, NY, USA, 2013.

[17] Unity Technologies, "Beginner Scripting - Unity Learn," [Online]. Available: https://learn.unity.com/project/beginner-gameplay-scripting?tab=overview. [Accessed 10 April 2020].