# GAME DEV TOOL GUIDE

THE 4 THINGS YOU NEED TO MAKE GAMES RIGHT NOW

BILAL ITANI | XENFINITY.NET

# Table of Contents

# Copyright Notice



# Introduction

Welcome to this quick guide on which tools to use for game development. Before we get into the details, I'll give you a briefing on the scope and purpose of this eBook, as well as how I hope it will benefit you.

I put this eBook together as not only a quick reference for the current optimal game development tools, but also to explain the fundamentals of what game development really consists of for beginners. The next sections boil down what each aspect or medium of games is at its most basic, and their contribution toward the combined experience that becomes a video game.

What this guide will **not** cover is any explicitly technical details, or any in-depth strategies to use the recommended software. Think of this as just a quick guide to give insight on how everything works, and which tools to use for the job.

With all of that said, we'll start with some fundamentals on the topic of game development.

# I. Defining Game Development

Before we tackle the tools of the trade, it's important to be clear about what we're trying to do. This is an important concept in programming logic because we often must find the best solution for a problem, which can be any resulting software. Imagine trying to create a sandwich when you don't know what goes between the bread. If we tried to create software, but we didn't know or understand the solution, we may as well be taking a guess at what goes in our sandwich. In our case, the most obvious problem is that we want to make a game, where the solution will be a completed game.

Now, this is an issue because our problem and solution are both very **vague**.

What we need to do now is **break down our problem** so we can get a better understanding of it, which is the main purpose of this eBook. Essentially, we'll be covering what game development and games truly are in this section. To begin with, we know that a game is an "**interactive medium,**" which basically means stuff goes in, and stuff comes back out. **User input** is the "stuff" that goes in, so controller axes, buttons, keyboard keys, mouse movement axes and others, and what comes out is dependent on the game, but usually includes **control** of an avatar or character, **audio** playback such as sound effects and music, and manipulation of **visuals** or graphics.

Based on this, we can classify a game as an **interactive medium in which a player is assigned control of an aspect or entity, and receives back information as visuals and audio**.

Now, since our problem is to create a game, which we just defined, to solve it, we must undergo **game development** to solve the problem, which will result in a completed game as a **solution**. Now before doing so, we must fully comprehend what game development is. Using our definition of a game, we can classify game development as **the process of creating logical flow and function, visual assets, and or audio assets to creative interactive software**. In this definition, I think it's important to say interactive software to avoid reusing the word game from the actual phrase.

Of course, people don't typically refer to each of the listed items this way. Logical flow and function is usually referred to as **code** or sometimes visual code, visual assets are usually referred to as **game art**, 2D art, 3D art, 3D models, sprites, and so on, and audio assets are of course usually referred to as **sound effects and/or music**.

Now that we have a proper understanding of games and their development process, let's get started on what we're actually going to need to go through this process.

# II. Programming Software or IDE

We begin with debatably the most defining factor of game development, which is the logical flow of the game. It can be considered as such because it tends to be the feature of a game that differentiates video games from other entertainment media such as films, which include no interactivity. It's important to understand that without any logical flow, a form of entertainment **cannot be played**, and therefore is not actually a game.

We can run through an analogy with a real world game, such as Checkers. In checkers, each player first picks a color, and then play and alternate turns until the game is over. We can define our game to have ended once one of the players is out of pieces due to the other player taking all of them. If we list everything out, we can see how this runs logically:

1. Allow player 1 to choose a color
2. Allow player 2 to choose a color
3. Set player 1 to be the current player
4. Keep doing the following until the game ends:
    a. Run the current player's turn
    b. Switch current player

Notice how our final numbered step says **until** something happens. This is our **game loop**. Every running game exists in a game loop. In a program, nothing can continue happening without a loop, so we know that we need a loop for **interactivity,** for example in the form of user input, and other constant logic. This is a very basic implementation of checkers, and of course doesn't take anything into account other than the very base logic. This is good because we want to leave other features out of scope for the design of our game logic. The rest of the content can be designed and implemented later.

Perfect! We just designed checkers. So what's the hard part? We need to consider how everything works. For example, what does it mean to choose a color? How does a player run a turn? What determines if the game ends? All of these questions are a bit beyond the scope of this eBook, but these are basically further **design questions**. Design should **always** be taken into account before implementation.

Okay, so now we've designed our program, and we want to implement it. How do we implement our developed logic? So far, every video game has been created using some form of **programming**, whether visual or traditional. If you don't know the difference between the two, visual programming languages are basically wrappers, or an **interface** to make it easier to use a traditional programming language under the hood. These are usually okay for simple games, but typically should be avoided for larger and more complex applications, especially when the developer needs more access to technical tools for subjects like **optimization**. That said, there are many new tools such as Unreal Engine 4's Blueprints system, which is a visual programming

language that can actually call and interact with functions and data from C++ and UE4 libraries, and can actually be used for very scalable projects, but I digress. Okay, let's move on to a brief list of tools we can actually use for this sort of thing.

1. **Visual Studio Community 2015 (C#/C++/Others)**
    - You can use Visual Studio for free, and it's the most commonly used IDE in Game Development now. Whether you're using C# or C++, which are both very common game development languages, Visual Studio is a robust option. Microsoft has now been working more closely with game development companies like Unity Technologies and Epic Games to develop tools for Visual Studio that with between Unity and Unreal Engine. For example, you can now use the debugger by attaching Visual Studio to your Unity project, and step through like a normal VS project. Visual Studio also ships with Unity now, so you can tell how they've been trying to push a workflow between the two applications.
2. **Xamarin Studio Community (C#)**
    - Xamarin Studio Community is also available completely free, and is designed specifically for C# using the Mono Framework, which is the same Framework Unity uses for its .NET implementation. This option is exemplary for Mac, as Visual Studio is not available for the platform, and Xcode, the go-to Mac IDE, is limited in comparison to both Xamarin and Visual Studio. Xamarin Studio is also based on the previous default Unity IDE, MonoDevelop, which I'll talk a bit about next.
3. **Honorable Mentions**
    - The following IDEs I'll talk about are either more limited or not quite specialized for Unity or Unreal development:
        i. **Xcode**
            - Xcode is the IDE that Mac developers typically use for any sort of development, but mostly for Mac and iOS using C, C++, Objective-C, and Swift. Many Unity developers prefer to use a different IDE for all the great C# and Unity tools. For example, I prefer to use Xamarin Studio when I'm using a Mac because it is completely specialized to work well with C#/.NET, which the Xamarin platform also uses.
        ii. **MonoDevelop**
            - MonoDevelop was previously Unity's default IDE that would ship with the engine installation up until Microsoft starting working in close cooperation with Unity to empower Visual Studio as a go-to environment for Unity developers with all of the traditional programming tools, like code completion and integrated debugging. MonoDevelop also had a tendency to crash and cause developers to lose all of their work. If you're planning to use MonoDevelop, I suggest using Visual Studio as an alternative on Windows, or Xamarin Studio on Mac.

# III. Game Development Engine

So this is the part that most people jump into, asking “What do I use to make a game?” Well, as you can probably tell, this is only one of the the software we actually need to create the full game, but depending on which engine you use, can be one of, if not the most, **impactful** tools on your workflow.

To make sure we actually understand what our choices affect, let’s go ahead and list what this piece of software actually does for us. A game engine by convention potentially contains the following:

1. A **library**, **framework**, **SDK**, or **API** intended for a certain programming language
2. Asset control features such as importing models, audio files, image files, and others
3. An **editor** software for visually managing a project

However, some will only include number 1, some will include 1 and 2, and some will include all three. OGRE3D and Irrlicht are two examples of an engine which is only a **library** or **API** without other tools like an **editor** or visual interface included. Unity, GameMaker, and Unreal Engine are engines which have all three of these features. Now before I recommend which engines to actually use, I want to crush some misconceptions from beginners about game engines.

1. **“Engine X isn’t scalable enough for my game idea.”**

You’re probably either wrong, or tackling a project too large for someone just starting out in Game Development. Oftentimes, new developers want to create an engine for their game, either for fun or because they think they need some extra scalability. The last thing you need is an extra 1-2 years from developing an engine when you already had the tools. Just scale your project down, or ask around to truly find out whether your project needs a custom engine.

2. **“I want to use Engine Y to create my game.”**

In general programming, new guys often want to choose a tool, and mend their project idea for it to fit that tool. What should be done is deciding which project idea to tackle next, and then determining which tools are best to develop that project, not the other way around. If you want to make your first game, make sure you know which game it is you want to make that fits your abilities, and then decide which tools will help you complete it best. It may sound silly, but it’s similar to the “Mac vs Windows” concept, where an Operating System is **just a tool** to help you get a job done, yet people still subscribe to only one blindly.

3. **“Engine Z’s features make it too limited, and therefore not a viable option.”**

I fell for this one before. Try to avoid labeling a tool for everything in general, as it may be useful for some projects, but problematic for others. Instead, figure out if this engine is a good fit for your specific project. If it is, compare it to your other choices and decide if you want to

use it. If it isn't, move on and use whichever other engine you think would be better. To reiterate, an engine is just a tool, not an ideology you have to subscribe to.

4. **"But I'm not a 'real' developer unless I use Technology X"**

Let me start this one off by saying it **doesn't matter.** The only thing "real" developers do different than "fake" developers, is getting games done and releasing them. No matter what technology you use, and how you've created your game, if it fulfills the vision you've created for it, and you've released the game, you've done what you were trying to do. No, you don't need to create a new game engine in C on a Linux machine to make your game. Again, you just need to decide what you're going to make, and which technology is going to be most efficient in helping you make it. Don't worry about what others are saying if you know that a certain engine is your best choice based on research.

With all that information out of the way, I'd like to start suggesting which engines are good for someone who is just getting started on the next page.

1. **Unity**
    - I always recommend starting off with Unity for beginners because it's easy enough to create full games, while allowing the user to learn anything by doing, and strengthening his or her programming skills. Game engines can sometimes have a lot of configuration in the way of getting started, but Unity bypasses this and lets new users focus on what they came to do: make games. A lot of that power comes from a **hybrid programming** system, where Object Oriented C# and Character Component Scripting are combined to allow for very organized, but easy to write code. With this combination, users can make games easily, but understand new programming concepts along the way. Along with Unity's programming system, you can also edit values using drag and drop functionality in the inspector, so **prototyping** game features can be very fast. There are also so many assets on the asset store that you can speed up the development process. Among those is **PlayMaker**, which allows for visual programming in Unity. My last point on why Unity being my #1 recommendation is that it supports builds for more platforms than any other engine, and it's easy to prepare projects for multiple platforms.
2. **Unreal Engine 4**
    - Unreal Engine 4 has the upper hand when it comes to making games look amazing with great performance. It's straightforward for **intermediate to advanced** programmers, and allows users to quickly build up all game features, with higher quality out of the box tools than Unity. However, there are a couple of problems with UE4 when you're just starting out. Before jumping into UE4, you'll want to have stronger programming skills and an understanding of C++. The **visual programming language** and system called Blueprints provided by UE4 allows you to perform advanced programming over other visual programming languages, but it does also have a higher learning curve than alternatives. The big takeaway here is that Unreal Engine is very robust and powerful, but if you're just starting out, you're going to have a larger learning curve than the other engines I mention here.
3. **GameMaker**
    - I personally don't use GameMaker for any projects, but I think it's a great starting point for anyone interested in quickly creating 2D video games without prior programming experience. GameMaker also provides users with a visual programming language, so you don't actually need to write any code to get your game running unless you want to. Similar to Unity, GameMaker also has drag and drop features, so you can develop games more visually and prototype them quickly.

# IV. Art or Graphical Software

Video game art is very diverse, so we need to actually decide what our art is going to consist of and look like. In almost every case, it's very important that all the art follows the same style for consistent visuals that can be enjoyed by the player. We have a few questions to ask ourselves to figure this out, such as:

- "Is my game going to be 2D or 3D?"
- "What kind of theme does my game follow?"
- "Do I want a very stylized look, or something more realistic?"

Once you figure out what all of these questions you have to answer are, you'll be able to figure out, if you haven't already, what your game is going to look like. Based on this, we can decide which software to use after we understand all the technical parts. I'd like to first make clear which software options there are, and what they're for.

1. **2D Graphic Software**
    a. **Raster or Pixel Graphics Software**
        i. This kind of software can be used to create art with a specified size and resolution, and is often used for User Interfaces and Web Graphics. The defining factors of Raster Graphics are that they are based on pixels with coordinates and color data, and therefore **cannot** be seamlessly scaled. Examples include **Adobe Photoshop** and the **GIMP**. If you're planning on digitally painting assets, painting software also go under this category.
    b. **Vector Graphics Software**
        i. The big difference between **Vector Graphics** and **Raster Graphics** is that Vector Graphics are based on mathematical equations rather than using coordinate data for pixels. Thus, vector graphics **can** be scaled and look the same in a larger context as they do in a smaller context. This allows you to easily resize and move your graphics in a redesigned context without any trouble, and can save you a lot of time. Examples include **Adobe Illustrator** and **Inkscape**.

So if you're making a 2D game, which of the two types should you use? I'll say first that you can get away with good looking visuals using either pixel or vector graphics, but you'll have an easier time if you can use vector graphics in the long run. However, you should expect to either be converting your vector graphic assets to pixels for game use, or be willing to purchase extensions for engines that allow for support of vector graphics for production use.

If you're using pixel graphics, I recommend using **Adobe Photoshop.** If you're not interested in spending money on building your first game, then I recommend starting out using the **GIMP.**

If you're using vector graphics, I recommend using **Adobe Illustrator**, but for a free option, try using **Inkscape**.

You also have the option of going 3D, but unless you're already skilled with the software I'll talk about next, it will take you longer than 2D. Therefore, I recommend starting with a 2D game, as you will be done much faster.

2. **3D Graphic Software**
    a. **3D Modeling and Animation Software**
        i. This kind of software is the standard for creating 3D characters and objects in video games. In 3D modeling software, the user can manipulate polygonal sides and vertices into desired shapes, such as characters and other 3D props. Included in the most commonly used 3D modeling software are animation tools that typically follow the same key frame based system, where animation data is stored as changes over multiple frames, and movement is interpolated between these frames to complete the animation. Examples include **Blender**, **3DS Max**, and **Maya**
    b. **3D Sculpting Software**
        i. Sculpting software is typically used by professional and enthusiast artists with the purpose of creating short films and animations. In the world of game design, sculpting is often used for concept art, or to demonstrate a high-quality version of a character or asset that will later be converted to a game ready version. Examples include **Zbrush** and **Sculptris.**

Before I recommend any of the 3D modeling software, I want to make clear that I only summarize sculpting software so that you don't have to worry about it. This is only something that would be used additionally in a big studio with the time and resources for long term and large scale concept art. To reiterate, it's most important that you actually get your game out there.

I recommend starting out using **Blender** because it's a very robust piece of software that's both free and open source, and used by so many artists. Because of this, not only is it very easy to customize and add features, but it's also so easy to get help and reuse solutions that others have provided. While Autodesk's **3DS Max** is the industry standard, it's much easier to get access to free and abundant knowledge of **Blender** when you're new.

# V. Audio Assets and Software

Just like visual art, video game audio is another crucial medium to direct your players' experiences. When deciding what your audio will sound like, you'll have to think about the theme of your game, whether it's casual or fast paced, and other features you might think will impact the feel of the game. However, audio is one of the assets that can often be outsourced or purchased as an asset pack without sacrificing fitting audio clips. Since sound effects can enhance any action or event in a game, before you get audio assets, you'll want to make sure you think of everything that you want sound effects for. It's also important to understand that different tools can be used for voice, miscellaneous effects, and music.

Let's start off with the scenario where you don't want to make your own audio, and I'll go through each of the three types used for video game audio one by one, starting with sound effects. The trick here is to look around for good sound packs, and try to see how they fit into your game. Browsing the Unity Asset Store and Audio Jungle are both great options. Personally, I've used music from Audio Jungle and SFX and Voice audio assets from the Unity Asset Store. If you want to have a pack that has perfect sound effects and voice assets for almost any small to medium size project, I recommend the Universal Sound FX pack which you can get directly from the website, or in the Unity Asset Store. As for music, I've had much better luck on Audio Jungle. Make sure you when you search for music, you listen to it and play it while testing your game to really understand how it will turn out with your game, and if it's the fit you're looking for, and that's about all there is to it.

Now, if you were looking to create your own audio, you'll have to take a couple of different pieces of software into account. Before you go for software though, you'll have to figure out which hardware is best for recording, and that's beyond the scope of this eBook. First, for both sound effects and voice recordings, Audacity is a great tool for audio manipulation. In Audacity, you can easily move and cut audio around, as well as apply effects like reverb, and it's totally free. If you're looking for something more advanced for sound effects, **FL Studio** is an option that is not only affordable, but also provides tools for producing music. If you're looking for more music specific software, I would do some research between the most popular products and decide which ones are for you.

# VI. The Final Verdict

Now that we've reviewed every essential area of game development, and we recognize that you'll probably have to wear a few different hats when producing your first game, we can decide all the software you should get right now so you can jump in head first into your game project.

1. Programming Software or IDE
    - **Windows**
        i. Visual Studio Community 2015 – (Website | Download)
            - Visual Studio is my number one recommendation for an IDE/Programming Software because it has the most support for Unity development, and Unity works with Microsoft to make it even better in that regard.
    - **Mac**
        i. Xamarin Studio Community – (Website | Download)
            - I recommend Xamarin Studio for Mac users because it has Mac support, includes many of the more robust IDE features like refactoring, and it handles opening Visual Studio solutions, or code bases and projects, just fine.
2. Game Development Engine
    - **Windows/Mac**
        i. Unity 5 – (Website | Download)
            - Unity is my primary recommendation as a Game Engine because of how well it allows both beginners and experts to expedite game development without sacrificing capability. New users can use a drag and drop system, while experienced users may be coding a robust library for an upcoming game. Unity programming is also performed in C#, which is another language that beginners can get into more easily than lower level languages with the right resources.
        ii. Unreal Engine 4 – (Website | Download)
            - Unreal Engine is my option number two as a Game Engine for beginners because while it has more robust features than Unity, including a much more advanced terrain system and higher quality stock shaders, it has a higher learning curve than Unity. This is due to a couple of design decisions, such as using C++ as the language for game scripting, and a lack of drag and drop features for most cases.
3. Art Software or Creative Suite
    - **Windows/Mac**

i. Adobe Photoshop CC – UI Features (Website | Purchase)
   - As defined in Section IV, Photoshop is a Raster Graphics based Software, which means quality is sacrificed upon rescaling. This is great for many situations, but can be replaced by Vector Graphics in many cases. I recommend Photoshop if you're experienced with it, or if you know exactly which sizes you're looking for on your projects, and only plan to scale them down.

ii. Adobe Illustrator CC – Sprites/UI Features (Website | Purchase)
   - Illustrator is Adobe's Vector Graphics based Software, and unlike Photoshop, the artwork created in Illustrator can be rescaled as desired due to being stored as equations rather than pixels. This is great for developing 2D sprites and UIs, especially when you're not completely sure what your game is going to look like in the end, as it gives you more room for experimentation.

iii. The GIMP – (Website | Download)
   - If you're not interested in spending anything on Photoshop or Illustrator for your game, then I recommend starting with the GIMP. The GIMP is a free and open source Software with similar base features to Photoshop intended for image manipulation, or editing images.

4. Audio Assets and Software
   - **Windows/Mac**

i. Universal Sound FX – (Purchase)
   - I highly recommend this package because it contains everything you're going to need for your first game, including SFX, Music Effects, and Voice Overs. Moreover, the outstanding creator has recently updated the package as of writing this eBook.

ii. Audacity – (Website | Download)
   - If you're planning on recording some voice overs, or editing some existing sound effects, Audacity is a powerful tool. In just seconds, you can change some effect, like pitch or reverb, on an audio track or record a new one, and export it to the file type you need.

# VII. Conclusion

Hopefully this list helps you decide which software tools to use when you work on your games. Before we finish, I'll mention a couple of choices I've left out, and why. For one, you'll notice that I've excluded 3D software from the Art Software list, and this is because, as I've mentioned before, it's generally easier to start out with a 2D game because of how much less there is for which to account, including the z axis, 3D animation tools, and some other features.

There are exceptions to this, like creating a 3D geometry mesh based game with no animations, versus a 2D game with complex animation and image effects. Of course, if you are more well versed in 3D art software than 2D art software, I recommend utilizing that skill.

I've also left out music composing software, as I believe it's better to outsource music for your first game, assuming you're not well versed in music composition. I say this because music composition is another large industry on its own, and it can slow down your development process by a longshot. That said, if you're experienced in creating music, or your game is based around it, then making your own music is perfectly reasonable.

That's going to be it for this guide, so I hope I've helped you start your game dev journey.

Here's to your first game,
Bilal