CHAPTER 11

Development Platforms, an Overview of Major Programming Languages, Engines, and Frameworks

Matthew Nolin, *1st Playable Productions, Troy, New York, U.S., mattnolin@1stplayable.com*

Key Summary Points

1. This chapter introduces the concept of programming languages, along with code, game engines and platforms, and how they are connected.

2. Also discussed are current programming languages and game engines and which might make the most sense to use depending on a project's goals.

3. Real world examples are provided to help connect all the information together

Key Terms

Programming languages
Code
Game engine
Development platform
Game platform
Unity
Android
iOS
C++
HTML5

Introduction

Digital games have evolved greatly since their creation half a century ago and so have the tools to make them. Originally there were no tools to aid game creators because the video-game field was entirely new. Today, one can easily find a wide assortment of programming languages, game engines and frameworks

to assist game developers in their trade. These languages, game engines and frameworks each have their own pros and cons to consider when deciding which ones to include in development. This chapter provides an introduction to these topics, information on currently popular game development tools, as well as a protocol for deciding which tools to use for your upcoming project.

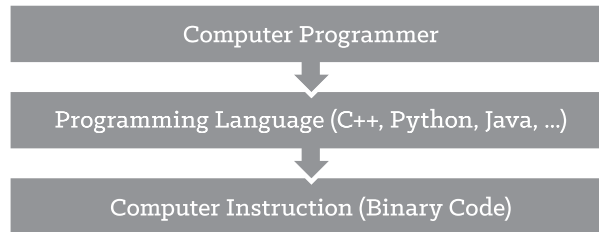

Figure 1. Code development process

Key Frameworks

Code: The Magic Behind Software

Ever wondered how videogames respond to your interactions? Code, a set of instructions for a computer, is what makes computers do something. Code makes computers react when you touch the screen, draw and animate images at a whim, it tells Siri how to respond to your request and so much more. Code is written in programming languages, such as Java and Objective C, which are used as a means to tell computers what to do. Programming languages are used because they are languages that computers are able to break down and understand. Tasks are relayed from the computer programmer, through the code that's written in a particular programming language, to the computer, which interprets the task into machine code that the computer understands.

There are many different programming languages, and much like spoken or written languages, each has evolved or been designed with different goals or usage in mind. Some, such as Python, are "high level," which means they abstract away many of the details of the underlying computations that are handled with each command or sentence. Programs written in these types of languages tend to be more readable and concise. Others, such as C and assembly language, which work "closer" to the underlying machine code, tend to produce more verbose and specific code, but can potentially run more efficiently and faster (Abelson, Sussman & Sussman, 1996; Carro et al., 2006).

Table 1. Overview of popular programming languages

Programming Languages	Overview
JavaScript & HTML5	A programming language most commonly used for developing websites, JavaScript can be considered the most popular programming language in the world (La, 2015). Although HTML5 itself is not a programming language, JavaScript is the language that most commonly takes advantage of features available in HTML5.
C++	C++ was designed to deliver the flexibility and efficiency of C while supporting object-oriented programming (Stroustrup, 1999). C++ is a very popular language for game development, particularly those that require a high level of performance (Williams, 2014).
Python	A higher-level language that is popular across the game and software industry. Python is often referred to as being easy to learn. Along with its straightforward nature Python is very powerful in a wide range of applications and can be used for many types of tasks (Ewing, 2014).
Objective-C	Objective-C is the language used for Apple's iOS and OS X systems and is a requirement at some point in the pipeline for developing on Apple devices. It is a superset of C while providing object-oriented capabilities and a dynamic runtime ("About Objective-C", 2014).
C	One of the earliest languages that is still widely used today, C is a popular language due to its high efficiency and portability. Several languages such as C++, C#, Objective C and many others have been influenced by C in some degree (tdammers, 2012).
C#	Developed by Microsoft, and first released in 2000 C# is intended to be a simple, modern, general-purpose, object-oriented programming language. Some of the primary goals of the language are to provide support for popular software engineering principles to encourage software robustness, durability and programmer productivity (ECMA, 2006).
Haxe	A more recent language that is part of the Haxe cross-platform toolkit. Developing with the Haxe programming language allows developers to easily produce cross-platform (Web/HTML5 and iOS/Android as an example) native code (Batchelor, 2015).
PHP	Similar to JavaScript, PHP is primarily used for the development of websites. One primary difference is that the language is run on the website's server, rather than on the user's computer they are using to view the website ("What is PHP?", n.d.).
Ruby	Ruby was developed to be an alternative to scripting languages such as Python and is often compared to it (Stewart, 2001). As such, it is also a high level language and one that is relatively easy to read and understand.
Java	A C-language derivative the Java language is second in popularity only to JavaScript (Perry, 2010; La, 2015). It's a higher-level language that was originally aimed at allowing game consoles and VCRs to communicate. One unique piece of information about Java is it is written to be run by a Java Virtual Machine (JVM), which means all one needs to run Java on a particular platform is a JVM for that platform ("The Java Programming Language", 1997).
Assembly	A very low level programming language, where the commands provided to the language directly correspond to the computer's machine code instructions. It is not very portable and does not have a lot of the conveniences of many modern languages. It can, however, result in the best performance of any programming language ("The Art of Assembly Language", n.d.).
Lua	Lua is a relatively simple programming language that has a light footprint and can be embedded into projects alongside other languages easily. It works on many computer systems as well making it an easy choice because of its portability. The design of Lua makes it easy for non-programmers to use (Slavin, 2013).
ActionScript	A close cousin of JavaScript, ActionScript is the language developed by Adobe for creating applications with Adobe Flash Player and Adobe AIR. It has been a very popular language for game development because of the ease of use of Flash although due to its lack of support on mobile devices it has dropped in use more recently (Williams, 2012; Cogswell, 2014).

Game Engines: What Makes a Game Tick

Game engines gather together a number of systems in one place to provide common functionality, such as certain types of animation, physics properties, sound effects, artificial intelligence, and more. The purpose of using a game engine is so that you do not need to create the entire game from scratch—some of the properties and functionality have already been provided. This greatly reduces development time, costs and the knowledge required to develop a game.

For high budget AAA games (a term used for games with the highest development and marketing budgets) such as *Call of Duty*, an engine is likely to have been specifically crafted with the target genre in mind. In some cases the game engine may even be developed only for that game. This is often expensive and time-consuming, but can allow developers to flexibly design their game as they are not limited by the constraints of any particular game engine. In some cases, game engines are available to be purchased or licensed from their creators. Because game engines are often designed for a particular purpose it is vital to choose an engine that aligns with your project goals, audience and design needs.

Table 2. Overview of popular game engines

Game Engine	Overview
Unity	Likely the most popular game engine at the moment, Unity (previously known as Unity3D) is a game engine combined with a very visual Integrated Development Environment (IDE) (Polsinelli, 2013). It allows developers to create games that run on most platforms from Apple and Android phones to the Xbox and PlayStation platforms.
GameSalad	Billing itself as "Amazingly easy to use" and aimed at creating games for iOS, Android and HTML5, GameSalad is a common choice amongst educators and students a-like (Defee, 2015). Although it does not have any flagship titles such as Angry Birds it does boast having had over 65,000+ games developed and 3 games that have been #1 in the US App Store ("Want to make games? GameSalad Makes It Easy", n.d.).
Source Engine	Developed by Valve Software, the Source Engine has been utilized to make a number of high profile PC games such as Half-Life 2, Counter-Strike: Source, The Stanley Parable and Garry's Mod. The updated Source 2 engine is aimed at increasing developer productivity (Kollar, 2015).
Unreal	The Unreal Engine is developed by Epic Games and is aimed for developers of all sorts, from student to indie to professional. It supports development of a variety of game types such as 2D, 3D and VR games on mobile devices, PCs and consoles ("What is Unreal Engine 4", n.d.).
In-house Engine	Some game development studios develop their own engines to suit their own needs, which are proprietary and not available for public use. Some examples of this are Electronic Art's Frostbite Engine, Konami's Fox Engine, id's id Tech 5, & Ubisoft's UbiArt Framework.
Construct 2	Targetted at indies, hobbyists, teachers and students as well as professional developers, Construct 2 allows rapid development of 2D style games and prototypes. No code is necessary; it allows anyone to build games ("What is Construct 2?" n.d.).
GamePress App	Create simple games without coding on the iPad with the ability to share them with friends. Aimed at introducing parents, teachers, artists and gamers to game development and allowing them to unleash their creativity ("Create the Games of Your Imagination." n.d.)

Game Engine	Overview
Scratch	A visual programming language and engine, Scratch works to make it easy for users to create interactive stories, animations, games, music and art and share them on the web. It was designed to facilitate the development of 21st century learning skill such as critical thinking, problem solving, communication, collaboration, creativity and innovation (Screawn, n.d.)
ARIS	Short for Augmeneted Reality and Interactive Storytelling, ARIS is an engine as well as an iPhone application that work together to create mobile, location-aware, narrative-centric interactive experiences (Gagnon, 2010). ARIS creates games where players experience a hybrid of virtual interactive characters, items and media placed in physical spaces ("ARIS" n.d.).
Cocos2d	A suite of frameworks that allow development of crossplatform games and apps. There are several forks that allow developers to create games in their favored language of C++, JavaScript, C#, Objective-C, or Python ("Cocos2d", n.d.).

Game engines may not include everything needed for a particular project, but are often extensible; middleware, frameworks, libraries, scripts, or other modules may be available to supplement or extend the available functionality. Middleware, frameworks, libraries and scripts are code that has already been written by a developer for a specific task or set of tasks. Using these common sets of code allows for more efficient development and for communities to arise around the use of them. For example, let's take cloth simulation. The physics and mathematics involved in making realistic clothing that interacts properly with the character and environment is highly complex, and middleware can hide most of that complexity from the game designer or programmer with carefully designed tools or interfaces. Cloth does not seem like it is so complex to design, but in a game environment, you need to describe exactly how something should move, or otherwise the game will not know to do anything with it. In this case, making clothing on a character respond to the characters' movements, such as a skirt reacting accordingly as the character moves their legs forward to walk, is a problem many games encounter and thus why middleware exists to solve this problem.

Platform: Almost Anything Can Run Games Now

There are a myriad of platforms available to develop games for: desktop computers, mobile phones and tablets, game consoles, and even children's toys. The decision of what platform to create games for can be a key factor in which programming languages and game engines should be used in development. To choose a platform or platforms to develop a game for, a number of questions need to be answered related to project goals, budget, team and so on. Is it the goal of the project to reach a population that's inside schools in the United States? If so, computer labs are currently ubiquitous across many schools, however the rate of technology progression over the last couple of decades has rendered much hardware in schools incapable of running any recently developed games. SMART Boards (interactive whiteboards) as well as smartphones and tablets are increasingly available in schools, although they have other complications that affect development, such as wireless availability and operating system differences. Understanding that these choices affect which engines, languages and even the team of programmers that can be used on a project is crucial.

According to the Joan Ganz Cooney Center's Teacher Attitudes about Digital Games in the Classroom (2012) report 85% of students can access digital games on PCs or Macs, while two-fifths can use interactive whiteboards and a quarter can access games via tablets. These numbers are moving quickly; tablets were introduced less than 5 years ago! Educators are learning about these games and platforms through a mix of their peers, self-interest and conferences. A new development to ease the burden on school districts to keep up with the ever-changing market is a trend referred to as "Bring Your Own Device" (BYOD). With seventy-seven percent of students owning cell phones and one third of those being smart phones BYOD seeks to utilize these devices in addition to the traditional computer labs (Richards et al., 2013).

It is easy to see how anyone can be overwhelmed with the amount of information required to have an informed discussion about game development in general, or especially educational games in particular. This chapter helps provide a baseline of knowledge to better assist.

Understanding Code

As mentioned above, code is what is written by programmers to help computers understand what they need to do. Each programming language is unique and thus code that is written in that language even if it is designed to do the same thing as code that is written in another language will likely be different. For example, the simplest task for a program to do is often just displaying text on the computer screen. When learning a new language, the first thing programmers will often do is take a "Hello World" tutorial. Figure 2 shows an example of such a tutorial in a few common programming languages.

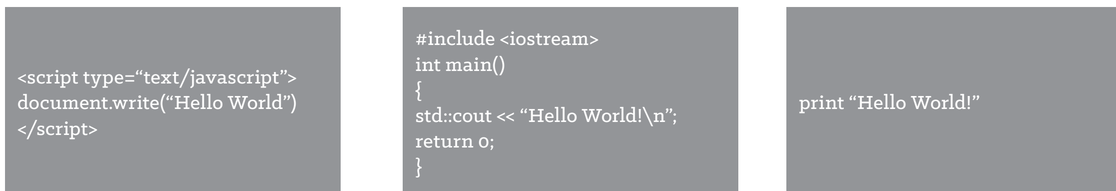

Figure 2. Code blocks to display "Hello World!" on-screen in JavaScript (left), C++ (middle), and Python (right).

The first on the left is JavaScript, and running it in your web browser would cause Hello World! to appear on-screen. The middle one is C++ and is a bit more complex because the programmer needs to include existing code written elsewhere to allow them to handle output to the display. Then inside of the "main" function, which is called when the code is run, it will print out "Hello World!" to the screen before completion. This is "lower level code" than JavaScript and is a bit more complex. Finally on the right we have another fairly straightforward example that is done in Python, which is the easiest to discern for any layperson.

Each of these code blocks require differing levels of complexity in the code itself in addition to being able to compile and test how they run. With JavaScript all one needs to do is create an .html file on their computer and open it in a browser such as Firefox to see if what they did worked as they wanted. C++ or Python requires other more complex setup on the computer ahead of time to be able to run the code the programmer has written.

Variables

Variables are containers of information for programming langauges that can be easily referenced. There are different kinds of variables and any specific programming language may have some but not others. Some simple examples are:

Boolean

A Boolean has a value of true or false. An example of a Boolean variable might be "gameIsOver." If a player had just finished the game then one could set the variable to be true with the following code (this example is from C++):

```
gameIsOver = true;
```

In this case, the "gameIsOver" Boolean would have to be initialized before it was used in this context. That would look like this:

```
bool gameIsOver = false;
```

Integer

An Integer holds a number value, which max size can change depending on the language and its usage. In C++ it is also important whether the Integer is signed or unsigned. Signed Integers support negative values where-as unsigned Integers do not. An example of the use of an Integer could be as follows:

```
int player1Score, player2Score, sum;
player1Score = 500;
player2Score = 1000;
sum = player1Score + player2Score;
```

For this example, 3 variables are created "player1Score," "player2Score" and "sum." "player1Score" is set to a value of 500 and "player2Score" is set to a value of 1000. The "sum" variable is then set to equal the amount of "player1Score" + "player2Score." In actual games "player1Score" and "player2Score" would be set to equal the amount they had scored during the actual game but for this example the values have been set directly.

String

Strings are variables that contain text. Depending on the language and the type of String that is utilized once a String is initialized and set it may not be able to be altered. An example of a String being used is:

```
std::string name = "Christopher";
std::string sentence = name + std::string(" is ") + std::to_string( name.length() ) + std::string("
characters long.");
```

In this case, we're creating a string with the name of a person "name" which is given the value "Christopher." Then we combine the name variable along with several other pieces to create this sentence "Christopher is 11 characters long." This is done by concatenating the initial name variable along with creating two short strings temporarily for the "is" component and the "characters long"

component. Then the "11" is created by using a function available called length() which returns the character length or size of the string it is run on. This is all put together as part of the variable "sentence" which can be used in the future to display this text all at once.

Logic

For code to react differently based on data from the game, programming langauges have conditional statements such as If-then-else statements and Switch statements. These popular conditional statements are a foundation of coding and are used widely. They allow for software to have appropriate actions based on what is occurring at any given time (as long as the programmer thought to plan for it and create a conditional statement for it). An example of this might be:

```
if lives < 1:
        gameOver()
```

In this example if there are no more lives left for the player to continue (measured by if the lives variable has a value of 0 or less) then the code will compute the gameOver() function. Another example showing a switch statement instead with JavaScript:

```
switch (event.keyCode) {
  case 37:
          leftArrowKeyPressed();
          break;
  case 38:
          upArrowKeyPressed();
          break;
  case 39:
          rightArrowKeyPressed();
          break;
  case 40:
          downArrowKeyPressed();
          break;
  default:
          notArrowKeyPressed();
          break;
}
```

In this case the JavaScript code will run the code in the appropriate function based on what specific keyboard button is pressed. This code could be used to move a character in a direction based on the keyboard input to the game.

Functions

Many programming languages utilize a feature called functions. Functions are a segment of code that can be called by name. In addition, certain pieces of information can be passed to the function so that the code can utilize that information. An example of a function might be one that updates a line of dialogue to use what the player has put in as their name. In python this might look like the following:

```
def displayWelcomeText(name):
  print("Welcome to our game " + name + ".")
```

This function could then get called in other areas of code and potentially even other functions. Functions are used to make it simpler to run pieces of code many times, rather than having to copy and paste the same code over and over again. Functions are also used to help keep code organized. If a lot is happening at once, functions can be used to logically split up separate parts of the code to make it so that you can still get a high level overview of what is happening without reading through all of the code. An example of this might look like this:

```
if hasBallCrossedGoal( ballLocation ):
        updateScore()
        playAudio("Goaaaaaal.mp3")
```

In this case a simple soccer game is checking if the ball has crossed the goal line via a function "hasBallCrossedGoal(ballLocation)". This function takes in the location of the ball (where it is in the game / on-screen) and then inside the function if we can guess by the name that it returns yes or no based on whether the ball is across the goalline. If it returns yes, then the game will run the function to "updateScore" and another function "playAudio("Goaaaaaal.mp3"). Based on the function names again we can make an educated guess that the game will then go on to play the "Goaaaaaal.mp3" audio file to the player(s). If someone wanted to make sure of what the functions are actually doing they would have to go to the source for the functions and see where they are defined. This is part of the reason why it can be important to have functions that are appropriately named.

Case Study One: C++

Letter Factory is an educational game targeted at early learners developed by 1st Playable Productions and published by LeapFrog. It is based on the popular DVD, also called Letter Factory. Because the game was published by LeapFrog, the game needed to run on both the Leapster Explorer and LeapPad mobile game platforms. Both systems require development to be done through LeapFrog's SDK (Software Development Kit). This also means that the game needed to be created in one of two supported languages: C++ or ActionScript. With an internal engine that was already widely used around the studio created in C++ The decision was made to adapt the engine to work with LeapFrog's SDK.

The decision to support C++ as the development environment provided a number of positive benefits. This allowed the vast array of knowledge that existed in the internal game studio engine to be brought over to this new platform with a low layer of code to work directly with LeapFrog's SDK. Any individual at 1st Playable Productions who had experience with most areas of the internal engine would see no difference in developing on this new platform, which would allow them to jump right in and be able to create at a similar level to other platforms. As the internal engine used by 1st Playable was already optimized for portable hardware such as the Nintendo DS, there were many areas that the engine performed very well and the team did not need to spend a large amount of additional time addressing performance issues.

Although many areas of the game were coded in C++ with the internal engine that existed, some areas needed to be extended to work within LeapFrog's SDK. This included areas dealing with the curricular, audio, and hint systems. New and unknown issues did come up when supporting the new SDK, as is the norm when learning new areas, however the use of the internal C++ game engine in this case helped reduce the number of new items for this project, which helped make the project successful.

Key Findings, Languages and Engines

This section details the programming languages and game engines in more detail.

Programming Languages

There are a fairly large number of programming languages out there in computer programming so we will focus on most popular ones at the moment. Languages often intersect within a specific game project, as teams and programmers play to the strengths and requirements of each platform and language. The following languages are used for a wide variety of games today: C++, Objective C, C#, Java, JavaScript and HTML5, Python and ActionScript.

C++ (Created in 1979): C++ is the language of choice for many games on consoles and handheld/mobile devices due to its flexibility, performance and ability to interact with the machine directly. C++ is a relatively complex language with a higher barrier to entrance than many other languages.

Objective C (Created in 1983): Objective C is the main language for interfacing with Apple's products, and has seen a tremendous growth in use out of necessity as the iPhone and iPad popularity have reached epic proportions. Objective C is required to interface with native iOS features such as using the camera within an application or being able to tell your game what to do when players tap the home button and exit the game (for example, you will probably want the game to save).

C# (Created in 2000): C#, which is pronounced as "C sharp," is a bit higher level than C++ and provides some additional conveniences, which can result in a lower learning curve and provide shorter development times. It is important to note that these conveniences can be problematic if programmers are attempting to implement very high performance applications and are unaware of how the language is handling things under the hood.

Java (Created in 1995): Java is often used as an entry-level programming language for students in college as it's more accessible than C languages but lower level than Python and web-based languages such as JavaScript. Java is one of the most popular programing languages and was popular for developing games on phones in the pre-smartphone era, but due to the need for high performance it has not made many inroads in being used widely in the game development community. Java is the core language for Android development, similar to Objective C's use in Apple's products.

JavaScript and HTML5 (JS Created in 1995, HTML5 is still in development): JavaScript has continued to get more powerful and faster as the Internet and its browsers have been updated throughout the last 10 years. It's now a powerful tool to create web-based games with. Due to the increases in performance of computers, browsers and the language itself in addition to the ability to use HTML5 for new multimedia features JavaScript and HTML5 are becoming widely used tools to create games with. HTML5 is very much still in development and not all of the same features are supported across all browsers which can make development challenging. Although they are newcomers into the space and their ability to run on any platforms (phones, computers, tablets) is promising, the performance level that is needed to run intricate games is still fairly unproven.

Python (Created in 1991): Used throughout game development and other areas of programming, Python is a more accessible language than most. It emphasizes ease of use and readability that make it very useful for a wide array of applications. In game development it can be used to create simple games with tools such as PyGame and is often utilized for scripting game events such as the AI responses and creating tools to assist more complicated game development projects.

ActionScript (Created in 1998): Although not the top choice for some developers at this point in its lifespan, ActionScript is still wildly popular due to the ease of creating games in Flash that can be readily available to anyone with a browser. Creating games with ActionScript and Flash still has a comparatively quick iteration time relative to other means but they are primarily limited to a PC audience.

Game Engines

Similar to programming languages, there are a number of game engines in use across game development today. These game engines are designed to allow developers to create games that fit their needs while working with a preexisting system that often times many engineers are familiar with. Many of the engines discussed below support games that can run on multiple platforms. To be able to use one engine to create games on all the major platforms saves a lot of development time and resources. This is wonderful for teams because members will not need to learn new tools as trends change and they can also be used across different teams within a game development studio without changing tools.

However, using a game engine, especially one that is new, can be a challenge if the game requires functionality outside of the engine. Working with an external game engine means your team may not be able to solve some problems that arise because of the engine itself. It is a fundamental problem that when you work with an externally developed game engine, the team must rely on it to keep the game running smooth and refraining from crashing. If there is a bug in the engine, or a feature that you are using in a unique way, it may require an engine upgrade or for the team to change its plans for development and use a different engine.

Unity: Currently at the top of the game development popularity list is Unity. The Unity Engine has been growing in its feature set and popularity due to the ease of working with it, along with its price point (it is free to publish on PC and mobile as long as you do not earn over a certain amount). As such, it is currently in use by students, indie game developers, and professional developers. Although relatively inviting when compared to most game engines, anyone unfamiliar with game engines will definitely need some time to learn the ropes. Happily, because of the popularity of Unity at the moment, there are many developers across the globe to lean on for tutorials and information to help in learning the Unity engine. Unity works with a wide range of current platforms such as Xbox 360, PlayStation 3, Xbox One, PlayStation 4, Wii U, iOS, Android, Windows Phone, PC, VR.

GameSalad: As far as ease of use, GameSalad is one of the most accessible engines to be able to get in and start creating things right away for just about anyone. The key feature about GameSalad is that everything is made through drag and drop behaviors and changing properties of objects to create games. There is no “code” that’s visible to the game creators, all of that is handled behind the scenes by GameSalad for the game creator. Although not viable for high end games it is a great tool to create simpler games and allow experimentation with the game design process. It also provides an opportunity for students (or anyone) to very easily create something and share it.

GameMaker: On the spectrum of accessible game engines, GameMaker is similar to GameSalad and is also relatively easy to grasp. Similar to GameSalad, it allows creators to drag and drop items and edit properties to create games, but the ability to edit the code directly furthers the ability to create unique games, as well as allow programmers or technical designers to create prototypes without relying solely on the drag and drop features.

Source Engine: Source is a fairly old engine at this point (originally developed by Valve Software for Half Life 2 in 2004), however it has been updated continuously and is available to be tweaked with every purchased copy of the game. Source is a robust engine with many tools within it to aid in development. It is a more complex engine but can create sophisticated games because of that complexity. Many mods have been developed based on the Source SDK and commercial games continue to be released for it such as commercial games continue to be released for it such as Titanfall (released in 2014). A much higher learning curve and expertise level is required for this type of game engine, but options like this do exist for anyone to be able to create their own game at a very professional level.

In-house Engine: Often times a game development studio may utilize an internally developed game engine or toolset to work on its games. This has a few benefits: team members within the company can be utilized on any project the company is working on, because they share the same engine and team members can get to the root of problems, since all of the code is available to work with. Some downsides are that this is that the engine is limited to whatever the company has worked on and without the sole purpose of the company being dedicated to the engine portions of it can become stale or out of date. It can also be costly as new additions to the engine and upkeep cost engineers who could otherwise be working primarily within existing engines to help create the game itself.

Case Study Two: Python

Through the development of a number of titles at 1st Playable Productions, we have amassed a large number of voiceover, special effects, and music files. We have a system to deal with this audio data, to more easily update, remove and change them within a game. The audio files are uploaded onto the web for us to download as they are created and updated by the audio team that we are working with. Unfortunately the tool does not notify users that audio files have been uploaded; rather, we have to rely on others to tell us that the files were uploaded or updated.

This is less than ideal in a few ways. First of all, many times the entire team does not have access to retrieve these files, which can sometimes create a bottleneck when the game audio needs to be updated and only one person has access to do so. In addition, it relies on individuals to make sure they download all the necessary files and place them in the appropriate location within the game.

This may seem like a small problem, but over time, small problems add up, and there are many places for human error. These types of issues are also always the ones that seem to be left off checklists, forgotten, or where time simply did not exist to do the process necessary. People on the team want to be focused on the game development, rather than worrying about where the audio files are located. This is the type of task that should be automated.

Since changing the overall system was not an immediate solution, 1st Playable developed a new tool using Python that provided information about when and where the audio files were uploaded or updated. In this case it was important to create a log that would show what specific files were downloaded, and when there was an error, so individuals could see that the file was supposed to have been created, but did not exist on the server yet. This tool is just one example of how Python could be utilized within a project's game development to help make the project and future projects more successful.

Future Needs

The trends of the future are leaning toward programming games to be played on more and more devices of all sizes and types. Whether it is the Microsoft Surface (imagine a coffee table touch screen computer that displays your photos when you place your phone on it), iPads, SMARTBoards, or the Oculus Rift (virtual reality headset), games are going to be developed for a wide range of experiences. These platforms all have unique design constraints, which require special approaches in how to develop games for them. However, we know that new tools will help assist with game development, engines and programming languages will be updated, and new ones will take shape. If one thing has become clear over the last decade, it is that game development is being democratized and that anyone can be a game maker now.

Case Study Three: HTML5

Released in 2009 on iOS by Rovio Entertainment, the game Angry Birds is one of the most popular of this generation. It has over 2 billion downloads across all of the available platforms through its regular and special editions. Rovio Entertainment has been expanding its brand by releasing the game on platforms outside of the original mobile market. One of these efforts was to release the game for web browsers with the idea that anyone with a recent Internet browser would be able to play the game. Rovio worked with Google to develop the game in HTML5 to make this happen.

They developed the game through the library PlayN with the Google Web Toolkit framework. Through this library they were able to program the game in Java and had the ability to use the cross-compiler to make the game available via Javascript (HTML5), ActionScript and Android. Although only the HTML5 output was used for this project. With the benefits of HTML5 the game could be played in the browser without players having to download anything to play it. Rovio did run into performance problems for later levels in which there are a large number of objects that collide and cause complex calculations, which they needed to work around. In addition, there is also an expectation that games load very quickly on the web, which they needed to overcome. Limited bandwidth was a part of this issue, even though it is behind the scenes to the player, the browser does need to download all the resources need to play the game. Another area that was troublesome is audio. To complete the game on time, the team needed to use Adobe Flash to allow the game to play sounds correctly. The close collaboration between Rovio and Google helped work through these challenges and they were successfully able to recreate the game in a couple of months for HTML5 output (Webber, 2011).

Best Practices

The following are best practices for choosing the right software, game engine, tools, or platform for your game.

1. **Research what is out there:** The field is always changing, and options that exist now did not exist only a few years ago. Options that were not appropriate then might be appropriate now. Technology is constantly changing, and new types of platforms and tools to make games are always coming out. Make sure you take a good look at the lay of the land before putting too much thought into any specific solution.
2. **Choose something updated recently & frequently:** A tool that does not have any recent updates or only gets updated infrequently is not going to likely be a good fit for your game. Support for OS (operating system) updates, new platforms, and bug fixes is something you want to see. Something that does not have these updates will likely hold back your development. You want a tool with long term viability so that the tool does not “go bust” during development.

3. **Choose something widely used:** A tool that is widely used is much more likely to have the community support that you will eventually need to help solve issues as they come up. A community of people who use the tool means it is also much more likely that people will come across similar problems and even share their solutions! If this is not an option for you, make sure you will have sufficient access to be able to get the support you will need to resolve any issues as they come up.
4. **Choose something that fits your cost needs:** This is obviously an important consideration. Different tools have different costs and licensing structures associated with them. For instance Unreal is free, but takes a percentage of revenue after you make a certain amount. Unity has an upfront cost for professional use but also has a free version. Certain tools may or may not be options for you because of this.
5. **Choose something that fits your team and project's experience, culture and needs:** Some tools are better at some things than others, so try to choose tools that excel at what your project aims to do and who is working on it. This seems simple, but it is important. In some cases things will align, and you'll have people on your team who are experienced with a particular tool and that tool happens to be perfect for what you want to build. Other times this may not be the case, you may have a team that's really experienced with a tool but you want to make a game for a platform that tool is just starting to support. In this case you'll have to weigh the development team's ability to adapt to a new tool and perhaps build a better experience, or decide whether sticking with the current tool would be a better option for you because you may actually be able to create something in the time that you have.

Resources

http://stackoverflow.com/
http://gamedev.stackexchange.com/
http://html5gameengine.com/
http://jquery.com/
http://www.python.org/
http://www.pygame.org/news.html
http://www.adobe.com/devnet/actionscript.html
http://unity3d.com/
http://gamesalad.com/
http://www.yoyogames.com/gamemaker/studio
http://source.valvesoftware.com/
http://box2d.org/
http://gamedev.net/
https://scratch.mit.edu/
http://arisgames.org/
http://cocos2d.org/
http://haxe.org/
https://code.org/
www.scirra.com/construct2

References

Abelson, Hal, Sussman, Jerry, & Sussman, Julie. (1996). *Structure and Interpretation of Computer Programs*. Cambridge, MA: MIT Press.

Apple Inc. (2014). About Objective-C. Retrieved from https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

ARIS – Mobile Learning Experiences. (n.d.). ARIS. Retrieved from http://arisgames.org/

Batchelor, James. (2015). *A technical guide to cross-platform development*. Retrieved from http://www.develop-online.net/interview/a-technical-guide-to-cross-platform-development/0202753

Carro, M., Morales, J., Muller, H., Puebla, G., & Hermenegildo, M. (2006). High-Level Languages for Small

Devices: A Case Study. *CASES '06 Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems,* 271-281. New York, NY: ACM.

Richards, J., Stebbins, L., & Moellering, K. (2013). *Games for a digital age: K-12 market map and investment analysis*. Retrieved from http://www.joanganzcooneycenter.org/wp-content/uploads/2013/01/glpc_gamesforadigitalage1.pdf

Cocos2d. (n.d.). Cocos2d. Retrieved from http://cocos2d.org/

Cogswell, Jeff. (2014). *5 Programming Languages Marked for Death*. Retrieved from http://insights.dice.com/2014/10/09/5-programming-languages-marked-for-death/

Defee, Andrew. (2015). *GameSalad Is For The Student And The Teacher [sxswedu]*. Retrieved from http://www.nibletz.com/startups/gamesalad-student-teacher/

ECMA International. (2006). ECMA-334: 4th Edition. *C# Language Specification*. Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Epic Games, Inc. (n.d.). What is Unreal Engine 4. Retrieved from https://www.unrealengine.com/what-is-unreal-engine-4

Ewing, Cris. (2014). *5 Reasons why Python is Powerful Enough for Google*. Retrieved from https://www.codefellows.org/blog/5-reasons-why-python-is-powerful-enough-for-google

Gagnon, David J. (2010). *ARIS: An open source platform for developing mobile learning experiences*. Retrieved from http://arisgames.org/wp-content/uploads/2011/04/ARIS-Gagnon-MS-Project.pdf

GamePress. (n.d.). Create the Games of Your Imagination. Retrieved from http://www.gamepressapp.com/

GameSalad. (n.d.). Want to make games? GameSalad Makes It Easy. Retrieved from http://gamesalad.com/

Kollar, Philip. (2015). *Valve announces Source 2 engine, free for developers*. Retrieved from http://www.polygon.com/2015/3/3/8145273/valve-source-2-announcement-free-developers

La, Alyson. (2015). *Language Trends on GitHub*. Retrieved from https://github.com/blog/2047-language-trends-on-github

Perry, Steven. (2010). *Introduction to Java programming, Part 1: Java language basics*. Retrieved from https://www.ibm.com/developerworks/java/tutorials/j-introtojava1/

PHP. (n.d.). What is PHP? Retrieved from http://php.net/manual/en/intro-whatis.php

Polsinelli, Pietro. (2013). *Why is Unity so popular for videogame development?* Retrieved from http://designagame.eu/2013/12/unity-popular-videogame-development/

Teacher Attitudes about Digital Games in the Classroom. (2012). Retrieved from http://www.joanganzcooneycenter.org/wp-content/uploads/2014/03/jgcc_vq_teacher_survey_2012.pdf

tdammers. (2012). *Why are several popular programming languages influenced by C?* Retrieved from http://programmers.stackexchange.com/a/135548

Scirra. (n.d.). What is Construct 2? Retrieved from https://www.scirra.com/construct2

Screawn, Julian. (n.d.). *ScratchProgramming.org: An Educator's Guide to Scratch Programming.* Retrieved from http://www.scratchprogramming.org/documents/ScratchProgrammingGuide.pdf

Slavin, Tim. (2013). *Lua.* Retrieved from https://www.kidscodecs.com/lua/

Stewart, Bruce. (2001). *An Interview with the Creator of Ruby.* Retrieved from http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html

Stroustrup, Bjarne. (1999). *An Overview of the C++ Programming Language.* Boca Raton, FL: CRC Press LLC. Retrieved from http://www.stroustrup.com/crc.pdf

University of Michigan. (1997). The Java Programming Language. Retrieved from http://groups.engin.umd.umich.edu/CIS/course.des/cis400/java/java.html

Webber, J. Angry Birds on HTML5, Accessed at http://www.infoq.com/presentations/Angry-Birds-on-HTML5 on October 24, 2015.

Williams, Glyn. (2014). *Why is C++ considered the best language for professional game development?* Retrieved from http://www.quora.com/Why-is-C++-considered-the-best-language-for-professional-game-development

Williams, Michael J. (2012). *How to Learn Flash and AS3 for Game Development.* Retrieved from http://gamedevelopment.tutsplus.com/articles/how-to-learn-flash-and-as3-for-game-development--gamedev-636

Yale. (n.d.). Foreward. The Art of Assembly Language. Retrieved from http://flint.cs.yale.edu/cs422/doc/art-of-asm/pdf/