

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.01 | 07/14/2021 | Christian J. Busca | Development of simultaneous team-based provisions. Develop a dataset to provide unique usernames to players to ensure one instance of the Game |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

The application in development is based on a four-round minute guessing game from the 1980's called, Win, Lose, or Draw. The Gaming Room client aspires to develop a game to be more immersive in experience, the client is developing their ideas into a game called Draw it or Lose it. The key selling point of this Game is the online multiplayer in which each user has a tag which is unique to that player, with multiplayer matches between teams. In order for this to be accomplished, they require a single instance of the Game with unique identifiers between each instances of the games, teams, and players.

## [Design Constraints](#_2et92p0)

The critical design constraint presented is enabling players to play Draw It or Lose It on multiple operating platforms such as PC(Windows, Mac, Linux), Mobile(Android./iOS), and Console(PlayStation, Xbox, Nintendo). That would be the most expansive way to distribute the Game and make the application cross-utilized. The client also needs a method of security through authentication to ensure the information provided is secure. Security and storage is a critical constraint for this project. The server will need to be able to provide images and graphics for 1000's of users at a time as well as managing the authentication of all users and bot prevention.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may require other projects. In addition, a logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

The classes provided within the program are all Entities. Game, Team, and Player classes are inherited from Entity due to the "is-a" relationship, referencing the UML diagram below you can depict the inheritance of each class. This is a superclass entity due to each class having common attributes between the ID and names. The Team class and Player class "has-a" relationship because the Team has Players assigned to the classification. This is similar to the Game "has-a" relationship with Team and GameService's role in organization of these games. IN reference to the chart below you can identify this as an Aggregation. A "has-a" relationship by definition is known as a composition, this means that an instance of one class has a reference to an instance of another class or an instance of the same class. You can see his in the UML since GameService references Game, which has a reference of Team, which has a reference for Player. A single GameService could potentially have multiple instances of Game, which can have multiple instances of Team, which has multiple instances of Player.

****

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client's needs and look at the situation holistically, as it all has to work together.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Cross-utilization between OSX and other operating platforms based in UNIX and Windows. Easy setup plug-and-play to minimize effort for setting up a server. Configuration options in the terminal as well as GUI. Expensive to license the products through Apple. A web-based application can be easily accessed through a web browser, which nearly all devices can use. If the Gaming Room wants an effective server on one machine, The cost of a Mac Pro is $6000, which is their high-end desktop.  Specs:  3.5GHz 8‑core Intel Xeon W processor, Turbo Boost up to 4.0GHz 32GB (4x8GB) of DDR4 ECC memory Radeon Pro 580X with 8GB of GDDR5 memory 256GB SSD storage  For the cost of this build compared to a machine running Linux or Windows.  [macOS Server](https://www.apple.com/macos/server/) is an option available to macOS users, and this software is also available on iOS, which would be a mobile device server. The product costs roughly **$20.** Profile Manager, which controls profiles and user's accounts from the desktop, or mobile device. Gives the host complete control of profiles, user accounts and passwords, and various other resources. Allows for remote push installs and wiping in the case a device is stolen. Apps are purchased in volume and distributed through the devices. Allows for the access of concurrent centralized data.  There are [other options available](https://www.macinstruct.com/tutorials/how-to-turn-your-mac-into-a-web-server/) that are cost-effective, such as Personal Web Sharing, creating a web address, and using DynDNS to forward that address to the site. | Terminal commands and a user interface give the host more options to configure the server more efficiently. A beginner-friendly GUI is available, but the power user can configure the server within a terminal. Most cost-effective option due to the open-source availability.  Linux is the most popular development environment for web-based applications. Development with Linux, Apache, MySQL & PHP, Perl, or Python is known as a [LAMP stack](https://www.ibm.com/cloud/learn/lamp-stack-explained) and is a culmination of all these languages to create a web-based application. Typically you would use a Linux distro as the OS, Apache as the web-server, MySQL as the database, and PHP as the language to script the site. Facebook, in its early years, utilized this stack as well as Wikipedia and WordPress.  Since the software is open-source ($0), the Game Room could utilize the stack without spending any money on the software, which leaves some wiggle room for maximizing their server build.  The most optimal setup would be to distribute resources across different functions rather than an all-in-one. Good practice would be to keep the server and database separate, which may also increase security.  On the hardware side, both Linux and Windows are equally matched since you can essentially build a PC and have one or the other, or even a dual boot system. However, there are many [builds online](https://pcpartpicker.com/forums/topic/320787-dedicated-gaming-server-build) for under $1500 (25% of the price of a Mac Pro) that outmatch the Mac Pro in terms of pricing and hardware capability. [Also this.](https://towardsdatascience.com/how-to-build-the-ultimate-data-science-pc-on-a-budget-69ee498dcdaa) | Executable binary developed for the sole purpose of hosting games on Windows system. .NET framework has plenty of documentation and tutorials available for those new to administrating on Windows. But much like OSX Windows is costly, but beginner friendly. Options for power users are available such as cmd and Powershell.  Much like Linux, Windows in terms of hardware out matches both the mobile and Mac OS platforms in terms of pricing and capability.  Also very similarly to Linux-based web development Windows offers the WAMP stack which is essentially Windows, Apache, MySQL, and PHP. The stack is plug and play, essentially you can run the [exe](https://www.wampserver.com/en/) and you have all the tools you need for your server. There are also options such as VS2019 IDE which carry all-in-one functions also has debuggers and compilers. The professional version must be licensed out and is suite to small teams.  Microsoft Azure is a useful tool to help deploy and build web applications. Options are available such as ASP.NET, Java, Node.js, php, and Python/MySQL. These guides are free to use and excellent source material for starting a server.  It costs $200 to buy a Windows 10 professional key from Microsoft and for the Gaming Room, since they are a business requiring licenses for a Microsoft product the subscription for Visual Studio Pro is $45 per month. | Least likely option, unless the admin is running a phone-farm. Which is a cluster of mobile devices paired to process commands in tandem. The server will have to have a permanent location and hardware capable of maintaining the traffic of multiple users and services. Power consumption is also critical.  This is the least likely way to host a server. It's completely possible to make a web server on Android, but the capabilities of a mobile device, even some of the high-end phones are not worth it.  Yes, this is a low-cost option but we are developing a server for a business so this would be unprofessional and extremely hard to maintain. |
| **Client Side** | The client requires knowledge and basic use of the Mac OSX platform. Most expensive option.  To ensure that the application works on Mac, you would need to test the web application in the Mac environment. Since it's a web-based application, this means running it in Safari and other web browsers on Mac such as Chrome and FireFox.  To develop on this platform, you would need at least one user experienced in Swift programming. In addition, graphic design and website building would be required, so having users experienced in HTML, CSS, and JavaScript is essential to play the Game within a web browser.  This requires a team of skilled developers with knowledge in multiple programming languages familiar with the Macintosh environment. I think having some users who know Android would also help as well, making the process of porting the code to the platform will be quicker and more efficient.  Since this web application is cross-platform touch support will be a consideration. | The client may require skills to run a command line and setting up the Game through this interface. The most cost-effective yet requires the most skill to build.  Developing on Linux would be more straightforward. Since Android is a Linux-based Kernel, it shares similarities between the platforms. The framework language for most applications written in Linux distros is under GNOME GTK ( C) or KDE (C++). At the same time, most applications on Android are written in Java. Linux distros are also capable of Java-Development and considering the nature of Java's cross-compatibility, I think the portability of this application will be less time-consuming compared to porting the application to mac OS.  Since this is a web-based application, the developers will need to know HTML, CSS, and JavaScript. Knowledge of the application usage on Android devices will prove to be efficient. Testing on different Linux devices/distros and web browsers will be necessary.  Since this web application is cross-platform, touch support will be a consideration. | Beginner-friendly, the individual setting up this server doesn't need to be tech-savvy. Windows isn't free and requires purchase for full control of the system.  Users knowing their way around Windows is key to developing a web based application on this platform.  Key ideas to hit on while client side development is occurring are having a team of familiar with the languages mentioned in the previous two brackets and having experience on this platform and Android.  Others considerations would be developing for multiple screen sizes, a lot of different devices can run Windows.  Also, making the application work offline would be a neat feature for testing purposes.  If the intent is to develop a Windows Application, this will need to be developed in C#, C++, or J#. This application can only be installed on windows. | A mobile application could provide the option to host or watch their server on the go. However, implementation on this platform could prove to be difficult.  Since the mobile application is natively Android, it allows the client to fine-tune their application's cross-compatibility with other platforms.  The main challenge would be porting the Android application to iOS, considering they already have the template for the mobile application. I think the most focus would be to make it as efficient as the Android version. You cannot simply run an Android application on iOS and vice-versa. Both platforms have different methods of processing the data.  Android utilizes JVM Bytecode while iOS uses compiled code, not to mention you cannot install an application on iOS unless it is on the app store.  This application cannot be ported, but it will have to be developed from scratch. Also cross capability between Android versions and iOS versions will have to be hit as well.  All devices are sized differently so you must account for each screen size on Android, but iPhones only have 5 different screen sizes from the most common devices.  Navigation and button mapping are also essential to the development between these two platforms. |
| **Development Tools** | Knowing multiple programming languages associated with web application development is crucial such as HTML/CSS/JS for front-end development as well as scripting such as Lua or Python. Java and other OOP languages would work well for game development.  Mac OSX uses Swift for applications developed within the platform. I'm not sure what IDE this platform uses, but if I remember correctly CodeWarrior was big for C/C++ development on Mac. I think Swift uses Xcode or Atom.  For development on Swift or Objective-C there are multiple options for IDEs available. CodeRunner is an excellent utility due to it's versatile environment it can run code in 25 different languages. It's $15 on the app store. There are other free IDEs to use out there such as Atom(Editor) and Visual Studio, as well as NetBeans.  It will be necessary to disperse the tasks associated with the development of this application on Mac. I think there should be at least one developer with an expertise in mac development and a team dedicated to constructed the webpage in HTML and CSS, then an accessible game in JS. | Knowing multiple programming languages associated with web application development is crucial such as HTML/CSS/JS for front-end development as well as scripting such as Lua or Python. Java and other OOP languages would work well for game development.  Linux has multiple tools available for development such as Visual Studio, Eclipse, Netbeans, Code::Blocks to name a few. There are plenty of options available and text editors include Vim, NP++, gedit, Atom etc. There is also Wine which enables Linux distros to run .exe from a windows installation.  Linux is virtually free and open source with all the software and libraries available to all users, making this the most cost effective option.  Knowledge of C++, C, and the shell will ensure the development is successful on this platform.  Multiple IDEs exist on Linux platforms and the options are nearly limitless.  Like I have said previously, there will need to be teams with specific scopes in this project such as one team focusing on the web page development. A team of users with experience in Linux development to guide the game developers on how to allocate all the resources for efficiency.  A team on the front-end development focusing on designing the web page and making the Game accessible and a back-end development team constructing the game service and network for the client. | Knowing multiple programming languages associated with web application development is crucial such as HTML/CSS/JS for front-end development as well as scripting such as Lua or Python. Java and other OOP languages would work well for game development.  Windows is nearly on par with Linux in terms of availability of tools. I think Windows has plenty of options available for development. Especially VS2019 and C# .NET framework. Most tools available for Linux are available for Windows as well. You also have the option to run virtual machines.  The cost of licensing out the IDE from Microsoft will be a monthly subscription which could end up costing the Game Room a lot of money as long as they are actively working on the Game.  There are free alternatives like NetBeans, Codelite, Eclipse, Xcode. But all IDEs have specific scopes rather than all-in-one functionality like VS Code or Visual Studio Pro.  Each process will require front-end and back-end development like on a Linux platform. Some individuals may be cross-utilized for both ends but ultimately the teams should consist of individuals who either have expertise in system administration, webpage development, or game development.  Should be noted that Windows is more vulnerable when it comes to cyber-attacks so a System Security Admin would be crucial. | Knowing multiple programming languages associated with web application development is crucial such as HTML/CSS/JS for front-end development as well as scripting such as Lua or Python. Java and other OOP languages would work well for game development.  Android, Swift, and Java are the most practical for mobile development. Depending on the mobile operating system, the ability to create applications and the rate at which they could be distributed are much quicker since nearly all users have a mobile device.  Hosting on this platform would not yield efficient results.  For the application development there should be a split into two teams.  Team Android will improve the application in it's current state and update it so current users will be able to play cross-platform.  Team iOS will focus on using the Android template and rewriting the application in the iOS environment. This also includes publishing the application to the app store. Knowledge of Swift and Objective-C may be required. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and recommend The Gaming Room. Specifically, address the following:

1. **Operating Platform**: Windows has perfect system administration tools and is arguably the most accessible and beginner-friendly software to be out on the market. Since there are more users on Windows, more data would be processed efficiently, and Windows has a plethora of libraries and packages that are plug-and-play, making the system super easy to use. Most software developed on this platform does not require further knowledge of the system and makes time for the user to learn the software associated with the project rather than learning 3 or 4 different skills such as compiling and building Lua to use for in-game scripting… for example. Instead, you can install the language and run it on your server. No further research is required.
2. **Operating Systems Architectures**: Windows graphical user interface is aesthetically pleasing to observe and makes all applications more accessible through the start menu or Search. Windows Explorer is available to search through multiple drives. PowerShell is available to superusers wanting to enhance their experience with Windows system administration.
3. **Storage Management**: The Windows server allows reallocation of information to solid-state storage to create more storage for a user's needs. The user can also manage their files and allocate them where they please, as well as being able to choose specific save locations, as well as being able to use the cloud to store saved data. Web services, messaging, graphics editing, and other services are included. The server will require terabytes of information. To play it safe, I think the server should be configured on a solid-state drive capable of holding at least 500 Gb. While the server storage takes place in a separate drive capable of storing at least 1 tb of information. This is all depending on the distribution of the Game, if the Game requires more storage, then it should be migrated and expanded. This information would need to be backed up, so external backup storage should be created.
4. **Memory Management**: At least 16Gb of DDR4 ram minimum. It would be best to run multiple instances of the Game while running a dev kit to ensure testing and game creation. In my opinion, your rig should be future-proofed at best, so you can maintain your server for at least 5 years before having to upgrade your system.
5. **Distributed Systems and Networks**: I think each option besides mobile operating systems can do what the other can do for the most part. Linux has more control of its integrated components, but that's the only benefit that I can see on the surface. There are multiple ways of publishing games on each platform. Unity seems to be an engine that works well on all devices. If the Game is being hosted cross-platform, we can use Minecraft's example of signing into a Microsoft account to play with other players using their accounts.
6. **Security**: Windows is probably the least secure of all the platforms, primarily due to user error or privacy inefficiencies. In recent years they have progressively gotten better, but many exploits are still occurring today. Malware, viruses, worm clusters, to name a few. Windows Defender is the default application, but there should be an extra installation of an antivirus like Malware Bytes for surety.