

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.

## [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 set-up plug-and-play to minimize effort for setting up a server. Configuration options in terminal as well as GUI. Expensive to license the products through Apple. | Terminal commands and a user interface which gives the host more options to configure the server more efficiently. A beginner friendly GUI is available, but the poweruser has options to configure the server within a terminal. Most cost-effective option due to the open-source availability. | 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. | 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. |
| **Client Side** | Client requires knowledge and basic use of the Mac OSX platform. Most expensive option. | 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. | 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. | 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. |
| **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. | 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. | 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. | 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. |

## Recommendations

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

1. **Operating Platform**: Windows has very good 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 which make 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 having to learn 3 or 4 different skills such as compiling and building Lua to use for in-game scripting… for example. Instead you can simply install the language, and run it on your server no further research 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 through 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. Obviously, this information would need to be backed up so an external back-up storage should be created.
4. **Memory Management**: At least 16gb of DDR4 ram minimum. You should be able 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 it’s integrated components, but that’s really the only benefit that I can see on the surface. There are multiple ways of publishing games on each platforms, 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 having to sign into a Microsoft account in order to play with other players using their accounts.
6. **Security**: Windows is probably the least secure of all the platforms, mostly due to user error or privacy inefficiencies. In recent years they have progressively gotten better, but a lot of exploits are still occurring today. Malware, viruses, worm clusters to name a few. Windows Defender is the default application, but for surety there should be an extra installation of an antivirus like Malware Bytes.