

Draw It or Lose It

# **CS 230 Project Three Software Design Template**

Version 3.0

## Table of Contents

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

[Table of Contents](#bookmark1)**2**

[Document Revision History](#bookmark2)**2**

[Executive Summary](#bookmark3)**3**

[Requirements](#bookmark4)**3**

[Design Constraints](#bookmark5)**3**

[System Architecture View](#bookmark6)**3**

[Domain Model](#bookmark7)**3**

[Evaluation](#bookmark8)**5**

[Recommendations](#bookmark9)**9**

## [Document Revision History](#bookmark10)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 08/23/25 | Cameron Hill | N/A |

**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

The Gaming Room wants to introduce their game “Draw It or Lose It” on more platforms, not just Android. In order to do that, they need an operating system that is reliable, flexible, and easy to work on, and it must be friendly with several different operating systems. This report explores the pros and cons of different options such as storage, memory usage, operating systems, and network capabilities. The jist of it is that Linux would be the best option since it meets all of the requirements aforementioned.

## Requirements

To grow “Draw It or Lose It”, The Gaming Room needs a platform that works on more than just Android. It should be easy to use, run smoothly across different systems, maintain user’s data security, and be scalable so the game can grow with the player base. On the technical side, it has to support object-oriented programming, handle a lot of users at once, and work well in a networked environment, such as online multiplayer. The platform also has to play nice with distributed systems (Android, iPhone, PC, etc..) so players can use it on different devices and still get the same experience.

## [Design Constraints](#bookmark11)

Since this is a web-based game running on multiple devices, we have to build everything to work in a distributed environment. That means the app has to run well on different operating systems and deal with varying network speeds and devices. Security needs, cross-platform compatibility, and system resources are also important considerations. These constraints will guide how the code is written, how data is stored, server capacity, and testing. If not done properly, we risk slow performance, bugs, or insecure user data.

## [System Architecture View](#bookmark12)

*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 be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.*

## [Domain Model](#bookmark13)

The UML diagram below shows how all of the moving parts of the game are intertwined and which objects are inherent to which. GameService runs the program and manages each of the Games, which have Team objects. Every Team has their own Player objects. They all inherit from the Entity class which holds common attributes such as ID and name. This model provides a solid framework which will make for easier design and updating.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## Evaluation

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 requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers are known for being reliable with strong security, so they can be a great option for software hosting. Unfortunately, the entry cost is high and with limited enterprise-level server adoption, it is the less practical choice. Hosting at scale would require dozens or hundreds of machines, and their hardware is far less modifiable than Linux or Windows. Often times, and entirely new suite of machines would have to be purchased if upgrading. Mac would not be the most cost efficient or flexible choice for hosting “Draw It Or Lose It”. | Linux is the industry standard when it comes to server side work because of it’s strong security, open-source nature, and customizability. It’s a popular choice for big game titles such as XCOM: Enemy Unknown and Rome: Total War. Linux supports C, C++, Java, and Python and development tools such as Eclipse, GCC, and Emacs, which solidifies it’s flexibility. For The Gaming Room, Linux would be a cost-effective choice to for hosting. | Windows Server is one of the most widely supported options. Tools include Visual Studio, .NET, and PowerShell. This server is easy to maintain and troubleshoot due to extensive documentation of support resources. It would take less time to get up and running because the environment is familiar to many developers. It can be more expensive than Linux, but still less than Mac. | While mobile devices are not used to host on the server side, they are important to consider for the end-state. In order to create mobile applications, development tools such as Android Studio and Xcode can be used, which will connect to the game server. Duplication of work won’t be an issue because Flutter, React Native, and Unity all allow developer teams to perform one build and then deploy to multiple platforms at once. It is crucial that The Gaming Room chooses a server setup than can properly support mobile devices since “Draw It Or Lose It” will be mainly played on them. |
|  |  |  |  |  |

| **Client Side** | The Apple ecosystem works incredibly in itself. Unfortunately, Windows absolutely dominates gaming with over 95% of Steam users, and Mac users making up 3%. Since most game studios don’t support their games running on Macs, it would be best to not try to re-invent the wheel by putting the extra effort for “Draw It Or Lose It” to run on Mac. They would have to rework the platform later, so everything from graphics and audio to anti-cheat and updates, for it to be offered on Mac. | Linux is safe and secure, and runs well on older hardware which makes it a good choice economically. At the same time, Linux’s toolset is modern with things like Vulkan for graphics, PipeWire or PulseAudio for sound, SDL2 for input and windows, and even Mesa for open-source graphics drivers. Cross-platform engines such as Unity or Unreal can produce Linux builds with low code drift which could benefit The Gaming Room, especially for updates. | Windows almost has a monopoly on the client market, so it is important to support those users. The interface is standardized and compatible with most controllers, webcams, and headsets. Unlike Mac, there is a wide gap between the lowest and highest-performing hardware. Windows users are all over the place in terms of computing power, so developers would have to optimize for the game’s performance across a broad spectrum or raise the bar of computing requirements which could alienate a lot of casual users. | This is the highest priority for a lot of applications nowadays since everyone has a mobile device on them at all times. Whether waiting for a table at a restaurant or flying in an Airplane, users could play the game on their phone. iOS performs strongly and has tight security, and Android still rules market share. One thing to keep in mind is that Apple’s App Store approval process is strict; for “Draw It Or Lose It”, the technical standards must be high. Battery and memory use are highly scrutinized. |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Tools** | Apple has a great development environment most centered around Xcode, including the Swift programming language and Interface Builder. The tools are stable, well-documented, and are designed to work specifically on Apple’s hardware. If developing for platforms outside of the Apple ecosystem, it can be tricky. Many popular game frameworks are not available on Mac which require users to work around that with things like Parallels or UTM, and those don’t always work well. This is the less practical option for The Gaming Room’s goal of bringing their game to different platforms. | Linux is a great option for developers in terms of flexibility, control, and access to open-source tools. It’s easy to manage frameworks with package managers like apt, dnf, and pacman. Developers can build their hearts out with tools such as Vulkan for graphics, PulseAudio for audio handling, and SDL2 for cross-platform input and output. Compatibility issues can arise because some commercial tools are not natively supported. The learning curve can be a lot, but it is great for skilled developers. | Windows has the most broad range of IDEs such as Visual Studio, Eclipse, Unity, and Unreal Engine, which makes it the dominant option for game developers. For high performance games, it offers native support for DirectX. Development setups can be heavier due to some tools requiring more configuration than linux such as Visual Studio’s extensive workloads, IIS’ GUI-heavy and spotty configuration, and custom configuration of SQL databases, services, firewalls, etc… This can slow down setup, but the payoff is great with Microsoft’s ecosystem. | The two big mobile platforms (iOS and Android) offer their own development environments to ensure apps run smoothly. iOS has Xcode and Swift which provide a polished toolkit, but the App Store’s approval process is a pain. Android development relies on Android Studio which is very flexible and easier to get apps on the Google Play Store or even third party app stores. There are also cross-platform tools like React native, Flutter, and Unity which enable developers to write the code once and deploy onto both platforms from there. Performance optimization is not on the same level as Mac or (most importantly) PC, but mobile platforms should be the highest priority for The Gaming Room. |

## Recommendations

1. **Operating Platform**: For The Gaming Room, the best option for hosting and expansion is Linux. It’s reliable, secure, cost-effective, and widely used for server environments. Unlike Mac, which is expensive and less flexible, or Windows, which comes with licensing costs, Linux gives us the stability and scalability we need without locking us into a proprietary ecosystem. Developers also know it well, so building and deploying updates will be faster. Another major advantage of Linux is its wide support for programming languages and tools. It works well with C++, Java, Python, and all the major engines developers need, while also offering strong community support if issues arise.
2. **Operating Systems Architectures**: The client-server architecture is the best fit for “Draw It Or Lose It”. Keeping the main game logic and data on the server means the clients (PC, mobile, etc.) only handle the interface and inputs. This separation makes the system more secure and easier to maintain because sensitive data and core logic will stay protected on the server. This will also reduce security issues and will make updates easier to apply. Using a layered or multi-tiered setup also keeps things organized, because the database, game logic, and UI can all be managed independently. If one part needs scaling or fixing, it doesn’t impact the whole system. This design gives us room to expand later because if new features like voice chat or achievements are added, they can have their own layer without disrupting the rest of the game.
3. **Storage Management**: Cloud storage is the most reliable option for this project. It gives us the ability to handle large files like images, while also storing player accounts, teams, and historical data in one place. Cloud providers also have built-in redundancy so data won’t be lost if one of the servers goes down. Since this game could be played across multiple devices, such as PC one minute and then switching to console, this setup ensures players can always access their progress without worrying about losing data locally. Another benefit is scalability; if the game suddenly sees a spike in traffic, the storage can grow automatically without major downtime. A real-world example of this is how Netflix uses Amazon Web Services (AWS) to store and deliver its massive video library. Even with millions of users streaming at the same time all across the world, the system scales to meet demand and prevents outages. The bottom line is that cloud storage gives “Draw It Or Lose It” room to grow without hitting a wall.
4. **Memory Management**: Linux is a strong choice for memory management because of it’s memory management tools like paging and caching. These help prevent crashes when RAM gets bogged down. The game will only load what is needed in the moment, such as the current image or active round information, instead of trying to keep everything in memory. This approach keeps the game fast and stable, even on lesser-advanced devices. It also helps when more players join, since resources aren’t wasted on things that aren’t needed. As the user base grows, good memory management keeps performance consistent no matter what hardware people are playing on.
5. **Distributed Systems and Networks**: “Draw It Or Lose It” will need to run smoothly across many different clients at once, which makes a distributed setup important. Real-time communication through APIs or WebSockets allows players to connect quickly without delays. Load balancing will spread out the traffic so no single server gets overwhelmed. If one server does fail, the others can pick up the slack, keeping the game reliable and preventing players from getting disconnected in the middle of a match.
6. **Security**: Strong security is the only option, especially when it comes to user accounts and data. All traffic between the app and the server should use HTTPS, and stored player data should be encrypted so it isn’t just sitting there waiting to be stolen. Logins should use something stronger than just a password, like OAuth or even multi-factor (as annoying as it can be) for extra safety. A recent example that shows how important this is was when Pokémon developer Game Freak got hacked in 2024 and had a ton of sensitive data leaked, including employee records and project assets (Polygon, 2024). That kind of breach not only hurts the company but also erodes players’ trust, and once that’s gone, it’s very hard to get back. To avoid ending up in the same boat, we’ll need to keep up with patches, do regular security checks, and use firewalls or similar tools to protect the backend.

Sources Cited

Welsh, O. (2024, October 14). *Pokémon developer Game Freak suffers massive data leak*. *Polygon*. https://www.polygon.com/news/465710/pokemon-game-freak-leak-hack/