

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/18/2025 | Juan Gonzalez | Initial evaluation and software design. |
| 2.0 | 10/04/2025 | Juan Gonzalez | Server, client and tool evaluation. |
| 3.0 | 10/18/2025 | Juan Gonzalez | Memory, storage, and security recommendation. |

**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 Gaming Room wants to expand Draw It or Lose It from an Android only app to a web based game that runs across multiple platforms. To achieve this, a software design is proposed that leverages object oriented principles and proven design patterns. A central GameService Singleton will coordinate all games, teams, and players, ensuring only one instance of the service exists in memory. Unique names will be enforced for games, teams, and players through iteration checks, which guarantees that duplicates are prevented. An Entity base class will unify common attributes (id, name) across all domain objects, simplifying code reuse and consistency. This design provides a scalable, secure, and efficient foundation for the client’s goal of supporting multiple teams and players while expanding beyond a mobile only platform.

## Requirements

* **Business Requirements:** Support multiple teams per game, multiple players per team, and ensure unique naming for all games and teams. Provide a central authority that enforces game rules and manages state.
* **Technical Requirements:** Implement a Singleton GameService to control application state, introduce Entity inheritance for consistent attributes, enforce uniqueness using iteration in add/find methods and provide a driver to test functionality.

## [Design Constraints](#_2et92p0)

The application must run in a web based distributed environment, which imposes requirements on scalability, concurrency, and networking. Only one GameService instance can exist at a time, requiring a Singleton implementation. Name uniqueness checks must be performed before new objects are added, requiring iteration through collections. Because this is a prototype, all data will be managed in memory, which resets at each run. For a full deployment, external storage and session handling would need to be added.

## [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 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](#_8h2ehzxfam4o)

The UML class diagram shows GameService managing a collection of Game objects. GameService provides creation methods (addGame) and retrieval methods (getGame) while enforcing the Singleton pattern so only one instance exists. Game objects encapsulate identifiers and names, fulfilling uniqueness requirements. This design demonstrates encapsulation (private fields with getters), abstraction (separating coordination from representation), inheritance (all domain objects extend Entity), and composition (games contain teams, teams contain players). The Singleton ensures central coordination, while iteration is used to enforce unique names and locate existing objects, meeting the client’s needs efficiently.**"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](#_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 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** | macOS is reliable and secure, but it is not commonly used as a server operating system in enterprise environments. While it can host small applications, it is tied to Apple hardware and has limited scalability for web based deployments. | Linux is the leading choice for server environments, known for its stability, scalability, and cost effectiveness. It supports a wide range of server software and databases, making it the strongest option for hosting. | Windows Server is widely used in enterprise systems and integrates well with Microsoft products. It is user friendly but requires licensing fees and often more system resources, making it less efficient and more expensive than Linux in large deployments. | Mobile devices cannot effectively serve as hosts for web based applications. Their hardware, operating systems, and networks are designed for clients, not for server hosting, so they are unsuitable for backend deployment. |
| **Client Side** | macOS provides a polished and secure client environment. It supports modern browsers and can run the game through a web client, but its user base is smaller compared to Windows. | Linux desktops are less common among end users, but they are capable of running the game through browsers. It offers flexibility and strong performance, but support costs may be higher since fewer users are familiar with Linux desktops. | Windows dominates the desktop market, making it the most accessible client environment. It supports all major browsers and development tools, ensuring wide compatibility for end users. | Mobile devices are critical platforms because they are widely used by consumers. Android and iOS apps or responsive web clients allow users to play the game anywhere, making mobile support essential. |
| **Development Tools** | macOS supports industry standard development tools like Eclipse, IntelliJ, and Xcode. It works well for Java and web development but is more expensive due to the cost of Apple hardware. | Linux has excellent support for open source development tools, including Eclipse, IntelliJ, and command line utilities. It is flexible, cost effective, and integrates easily with servers, making it ideal for both development and deployment. | Windows supports a wide variety of IDEs and tools, including Eclipse, IntelliJ, and Visual Studio. It is user friendly and widely used, but it may require licensing costs and tends to use more system resources. | Mobile development requires platform specific tools such as Android Studio for Android and Xcode for iOS. While powerful, development is fragmented between platforms, requiring more time and expertise to ensure full coverage. |

## 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**: The most appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It is Linux. Linux is lightweight, stable, and efficient, making it a strong choice for running Java applications with consistent performance across different environments.
2. **Operating Systems Architectures**: Linux is an open source operating system that supports 64-bit processing. It provides strong performance, system stability, and flexibility for running applications. It is also commonly used for hosting servers, making it a strong choice for future expansion.
3. **Storage Management**: Linux uses file systems such as ext4 that are efficient and dependable for storing files. For this prototype, storage will be managed in memory using Java collections, but Linux also integrates well with databases if the game grows in complexity.
4. **Memory Management**: The JVM on Linux manages memory automatically, handling allocation and garbage collection. This ensures that game objects such as teams and players are efficiently managed in memory.
5. **Distributed Systems and Networks**: While this prototype runs locally, Linux has excellent networking capabilities. If the game later expands to a multi-user networked version, Linux can easily support distributed systems and client server communication.
6. **Security**: Linux is well known for its security features, such as user permissions, firewall tools, and frequent updates. Combined with validation in the Java application itself (like enforcing unique names), this provides a secure foundation for the project.

**Project Two**

**Server Side:**

macOS, while secure and stable, is not commonly used as a server platform in enterprise deployments. Its reliance on Apple hardware makes it costly to maintain, and it has limited scalability options compared to Linux and Windows. macOS could host small internal applications or prototypes, but it is not well suited for production server environments where distributed scaling and performance are required. The cost of Apple hardware also makes it impractical for large scale hosting.

Linux is the leading choice for hosting web applications in modern distributed environments. It is an open source platform known for its stability, security, and scalability. Linux servers are widely supported across cloud providers and integrate seamlessly with common backend technologies such as Apache, NGINX, Node.js, and Java application servers. Because Linux itself is free, there are no direct licensing costs, and the client only pays for cloud hosting or optional vendor support. For the Gaming Room, Linux offers the most cost effective and reliable option for supporting thousands of players with the ability to scale quickly using containerization and orchestration tools.

Windows Server is a widely adopted enterprise solution, especially for organizations that rely on Microsoft technologies such as Active Directory, IIS, and .NET frameworks. It is user friendly, integrates well with other Microsoft products, and benefits from a large support ecosystem. However, Windows Server requires licensing fees, which can increase costs, and it generally consumes more system resources than Linux. For the Gaming Room, Windows could be a viable option if the development team leans heavily on Microsoft technologies, but otherwise, Linux would be the stronger, cheaper choice.

Mobile platforms such as iOS and Android cannot serve as backend hosts due to their hardware and OS limitations. These platforms are not capable of handling large scale server operations. Therefore, they will not be considered as hosting options for the web based application.

**Client Side:**

macOS provides a polished, secure, and user friendly client environment. Although its market share is smaller than Windows, macOS has a loyal user base, and it supports all major browsers, making it an important platform for reaching Apple desktop users. Ensuring compatibility with Safari and Chrome on macOS will provide accessibility to this group of players.

Linux desktops are less common among general users but are capable of running the game application through modern browsers. Linux provides strong performance and flexibility, but its smaller user base means it will not be the primary client environment. Still, ensuring browser compatibility on Linux adds value by making the game accessible to technical users who prefer open source systems.

Windows dominates the desktop client market, making it the most important traditional platform to support. Virtually all major browsers are available on Windows, and it is the most familiar environment for the majority of end users. Supporting the game on Windows ensures that the largest number of desktop players can access the game seamlessly.

Mobile devices represent one of the most critical client platforms for the Gaming Room. Android and iOS collectively account for the vast majority of consumer usage worldwide. Developing the game as a responsive web application ensures that players can access it from any device using a browser. Additionally, providing support for mobile specific features through Progressive Web Apps (PWAs) or optional native wrappers like React Native or Flutter, can enhance the mobile experience. This ensures that the game reaches the broadest possible audience.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Platform** | **Characteristics** | **Advantages** | **Weaknesses** | **Server Deployment** | **Licensing Cost** |
| **Mac** | Secure, polished, tied to Apple hardware | Greate developer tools, essential for iOS builds. | Not scalable for servers, expensive hardware. | Not suitable for production servers. | Bundled with costly Apple devices. |
| **Linux** | Open source, widely used for web hosting | Stable, scalable, cost effective, strong community | Requires technical expertise, distro variety may add complexity | Excellent choice for hosting and scaling web apps | Free OS, optional support costs. |
| **Windows** | Enterprise ready, Microsoft ecosystem integration | Familiar tools, wide enterprise adoption | Licensing fees, higher resource usage | Good for .NET/IIS environments, less efficient for web scale. | Windows Server + CAL licensing costs. |
| **Mobile** | Client only platforms, iOS/Android | Massive reach, PWA/native wrappers | Device/OS fragmentation, app store policies | Can’t host backend servers | Free dev tools. Hardware and annual fee from Apple |

**Development Tools:**

macOS supports many of the same development tools as Linux, such as Eclipse and IntelliJ, but also includes Xcode, which is essential for developing and signing iOS applications. This makes macOS a necessary platform for building and testing the iOS version of the game. However, the reliance on Apple hardware increases costs for the development team, as Apple devices must be purchased and maintained.

On Linux, development is supported by a wide range of open source tools such as Eclipse, IntelliJ IDEA, and command line utilities. These tools integrate well with server environments and allow teams to build, test, and deploy applications efficiently. Since Linux is open source, there are no direct licensing costs, which keeps expenses low and makes it highly attractive for the Gaming Room’s development pipeline.

Windows provides access to Eclipse, IntelliJ, and Visual Studio, the latter of which is particularly powerful for developing applications in the .NET ecosystem. It is a user friendly environment familiar to many developers, but licensing fees may apply for enterprise editions of Visual Studio and for Windows Server deployments. While development on Windows is fully viable, it tends to be more costly than Linux.

Mobile development requires platform specific tools. Android applications are built using Android Studio, while iOS applications require Xcode. If the game is developed as a Progressive Web App, the bulk of the development will take place in web focused environments such as VS Code or IntelliJ, with Android Studio and Xcode only used for testing and packaging. While these tools are free, the challenge lies in the expertise and time required to ensure compatibility across both platforms. Additionally, publishing to app stores may incur fees, such as Apple’s annual developer program cost.

**PROJECT THREE**

**Operating Platform:**

Linux is a monolithic kernel operating system that supports both 32 and 64-bit architectures. Its design allows the kernel to manage core functions such as device drivers, memory management, and system calls efficiently within one cohesive structure. For a server environment, the 64-bit architecture is preferred because it provides better multitasking capabilities, memory addressing, and performance under heavy loads. Linux also supports modular extensions, meaning new functionality such as networking protocols or database modules can be loaded dynamically without restarting the system. This modular architecture ensures high uptime, a crucial requirement for continuous online multiplayer games like *Draw It or Lose It*.

**Storage Management:**

For storage management, the Linux ext4 file system is recommended due to its reliability, journaling capabilities, and high performance under concurrent read/write operations. Ext4 minimizes data corruption risk and supports very large file sizes, making it suitable for managing player data, game states, and session logs. While the prototype currently stores data in memory using Java collections, the production version can integrate relational databases like MySQL or PostgreSQL, or NoSQL solutions like MongoDB to store user profiles and gameplay records persistently. Linux natively supports mounting and managing multiple volumes, allowing seamless scalability as data requirements grow.

**Memory Management:**

Linux uses a virtual memory management system that allocates and reclaims memory dynamically. It combines paging and swapping techniques to ensure efficient use of RAM while preventing processes from overconsuming memory resources. The Java Virtual Machine (JVM) on Linux adds another layer of intelligent memory management through automatic garbage collection, which identifies and removes unused objects. This layered memory management ensures that *Draw It or Lose It* can handle multiple active games and users simultaneously without memory leaks or performance degradation.

**Distributed Systems and Networks:**

To allow *Draw It or Lose It* to communicate between various platforms, the system can adopt a distributed client server architecture built on web services or RESTful APIs. The Linux server would act as the central hub, managing authentication, player sessions, and game synchronization, while clients on Windows, macOS, Android, and iOS connect through secure network protocols like HTTPS or WebSockets. In case of connectivity loss, Linux supports fault tolerant configurations using load balancers, redundancy, and clustering. This design allows game instances to synchronize players actions and data across devices, maintaining consistency even in distributed environments. Linux’s robust network stack, like SSH or IPv6, provides reliable data transmission and monitoring tools to quickly address outages or latency issues.

**Security:**

Security is a fundamental requirement for The Gaming Room. Linux provides multi-layered protection mechanisms, including user and group permissions, SELinux, and built in firewalls such as iptables and ufw. For data in transit, SSL/TLS encryption can be enforced on all network communications, ensuring player data and login credentials remain secure between client and server. On the application level, the Java software can implement additional safeguards like input validation, toke based authentication, and secure session handling. The combination of Linux’s kernel level protections and Java’s security frameworks provides strong defense against unauthorized access, data breaches, and common web vulnerabilities such as injection or cross site scripting.

By deploying *Draw It or Lose It* on a Linux based distributed architecture, The Gaming Room gains scalability, performance, and costs efficiency. Linux’s open source flexibility, superior memory management, and robust networking stack create a solid foundation for future multiplayer growth, while built in security mechanisms ensure user protection across all connected platforms.