

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 |
| --- | --- | --- | --- |
| 1.3 | 08/17/2025 | Jim Misel | Finalized recommendations section |

**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 is transitioning its Android-only game, Draw It or Lose It, to a web-based platform that can support multiple client environments. This document outlines a software design solution that meets the client’s core requirements, including support for multiple teams and players, name uniqueness, and a singleton game service architecture. Object-oriented programming principles such as inheritance, encapsulation, and design patterns like Singleton and Iterator were applied to ensure efficient and scalable development. The proposed design enables clear entity management, reusable code, and the flexibility needed for future enhancements.

## Requirements

The Gaming Room requires that Draw It or Lose It transition from an Android-only application to a web-based solution that supports multiple platforms, including Windows, macOS, Linux, and mobile devices. The system must allow multiple teams and players to participate in games concurrently while ensuring that team, game, and player names remain unique. A single instance of the game service must be maintained in memory at all times to prevent duplication and data integrity issues. The design must be scalable to handle growth in user demand, support responsive performance across platforms, and provide strong security for user data and communication.

## [Design Constraints](#_2et92p0)

The key design constraints include supporting a distributed, web-based environment and ensuring that only one instance of the game service exists in memory (Singleton). Additionally, game, team, and player names must remain unique, necessitating the use of an Iterator pattern for searching existing names. These constraints influence the application to be stateless between clients while maintaining centralized control over data integrity, requiring clear class design and appropriate use of collections and access control.

## [System Architecture View](#_ilbxbyevv6b6)

The application will follow a client-server architecture. A centralized Linux-based server will host the game logic, data storage, and communication services, while clients on Windows, macOS, Linux, and mobile devices will connect through web browsers or lightweight apps. Communication between clients and the server will be handled through RESTful APIs and WebSockets, ensuring scalability and real-time responsiveness in a distributed environment.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram shows a centralized GameService class managing a collection of Game objects. The Game class, along with Player and Team, now extends a shared Entity base class that contains common fields such as id and name. Object-oriented principles like inheritance and encapsulation are clearly demonstrated. The Singleton pattern is used in GameService to ensure a single instance manages all game objects. The Iterator pattern is used to enforce name uniqueness by searching collections before adding new instances. These patterns contribute to a flexible and efficient system design.

**"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 can run server software like Apache or Nginx but is rarely used in production. - Better suited for development than hosting. **Licensing cost:** Requires Apple hardware (high upfront cost). | Fully supports server-based deployment. - Ideal for hosting web apps using Apache, Nginx, or cloud servers. - Highly scalable and widely used for enterprise hosting. **Licensing cost:** Free (open-source). | Supports hosting via IIS or third-party web servers. - Strong enterprise support and integration with Microsoft services. **Licensing cost:** Windows Server requires paid license (Standard ~$1,000+). | Server hosting not done on mobile; clients connect to backend web server hosted on Linux/Windows. |
| **Client Side** | Full compatibility via Safari, Chrome, Firefox browsers. - No additional cost beyond hardware. | Users access the game via any modern web browser (Chrome, Firefox). - No OS-specific issues for browser-based delivery. | Compatible with major browsers (Edge, Chrome, Firefox). - Easy desktop integration for users already on Windows. | Access through responsive web app in mobile browsers (Chrome, Safari). - Optional native app requires separate |
| **Development Tools** | Languages: Java, Swift (for native), JavaScript for web. - IDEs: Xcode (free for macOS), VS Code. **Cost:** Free IDEs; Apple hardware is expensive. | Languages: Java, Python, JavaScript, HTML5, CSS. - IDEs: Eclipse, IntelliJ IDEA (Community), VS Code. **Cost:** Most tools are free/open-source. | Languages: C#, Java, JavaScript. - IDEs: Visual Studio (Community free, Pro/Enterprise paid). **Cost:** Some IDE features require licenses; Windows OS license also required. | Languages: Java/Kotlin (Android), Swift (iOS), plus HTML5/JS for web. - Tools: Android Studio (free), Xcode (free). **Cost:** Apple Developer account $99/year for iOS apps. |

## 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 recommended operating platform for hosting *Draw It or Lose It* is Linux. Linux is open-source, cost-free, and widely adopted in the industry for hosting scalable web-based applications. Its proven stability, high performance, and security features make it the ideal choice for The Gaming Room. Choosing Linux also avoids licensing fees associated with proprietary platforms, helping the client manage costs while ensuring long-term reliability.
  2. **Operating Systems Architectures**: Linux supports a modular, multi-user, multitasking architecture that enables concurrency, process isolation, and efficient thread management. These architectural strengths are well suited for handling the simultaneous activity of many players in a distributed gaming environment. For Draw It or Lose It, this ensures that concurrent matches can run smoothly without processes interfering with each other.
  3. **Storage Management:** The application requires managing approximately 1.6 GB of high-definition images along with user profiles, game states, and settings. Linux file systems such as ext4 or XFS provide journaling for crash recovery and efficient handling of large files, ensuring reliable storage of these assets. To further improve performance, only essential images can be stored locally while the full library resides on a remote server, supported by a content delivery network (CDN) for rapid asset delivery regardless of player location. Compression techniques may also be applied to reduce storage size while balancing the processing cost of decompression during gameplay.
  4. **Memory Management**: To ensure responsiveness during gameplay, memory management techniques must be carefully implemented. Linux uses demand paging, virtual memory, and kernel-level swapping, which help allocate memory efficiently between multiple processes. Within the application, techniques such as lazy loading (loading images only when needed), caching (keeping recently used images readily available), and preloading the next image before display can significantly reduce latency and improve user experience. Optimized image formats and memory compression will help the game run effectively even on platforms with limited RAM.
  5. **Distributed Systems and Networks**: Draw It or Lose It must support players across multiple platforms and devices. This can be accomplished using RESTful APIs for standard data exchange and WebSockets for real-time communication during gameplay. By hosting the system on cloud platforms such as AWS, Azure, or Google Cloud, the game can take advantage of load balancing, redundancy, and auto-scaling, ensuring continuous availability even during spikes in user activity or partial outages. This distributed design will support the client’s need for scalability and reliability.
  6. **Security**: Protecting user information is a critical requirement. Linux offers strong built-in security features, including user privilege separation, firewalls (iptables), and SELinux policies for access control. All communication between clients and servers should use TLS encryption over HTTPS, ensuring data confidentiality and integrity. Additional safeguards such as OAuth-based authentication, secure API gateways, and encrypted storage for sensitive data will further protect user accounts and maintain trust across all platforms.

By leveraging Linux’s stable and secure architecture, modern file systems, advanced memory management, and distributed network capabilities, The Gaming Room will be able to scale *Draw It or Lose It* effectively to multiple platforms. These recommendations provide a strong foundation for performance, security, and long-term growth.