

# Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design** 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 | 11/07 | Nathaniel Gratton | Implemented Entity, addPlayer, addTeam and updated inheritance of Game, Player, and Team. |
| 1.1 | 12/06 | Nathaniel Gratton | Reworked the Recommendations to better outline the future of the game |

## [Executive Summary](#_sbfa50wo7nsh)

In this software design document, we address the challenge of transforming The Gaming Room's existing Android application, "Draw It or Lose It," into a versatile, web-based game application. Our solution leverages a distributed web architecture to make the game accessible across various platforms, including desktop and mobile devices. This approach not only broadens the game's reach but also enhances user engagement by facilitating cross-platform compatibility.

## Requirements

The primary business requirement is to extend the accessibility of "Draw It or Lose It" beyond its current Android-only availability. Technically, the game must support multiple teams and players, ensure unique game and team names, and maintain a single instance in memory to manage game states effectively. The web-based solution must be scalable, reliable, and maintain a consistent user experience across different platforms.

## [Design Constraints](#_2et92p0)

Developing a web-based game application introduces several constraints, including browser compatibility, network reliability, and responsive design for various devices. The need for real-time updates and low latency is crucial for a seamless gaming experience. Additionally, ensuring data consistency and managing state across distributed systems pose significant challenges.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram for "Draw It or Lose It" includes classes such as GameService, Game, Team, and Player, all inheriting from a base Entity class. This design demonstrates the principle of inheritance, promoting code reusability and scalability. The use of the singleton pattern in GameService ensures a single point of control and coordination for game instances, aligning with the requirement of a single memory instance.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers offer robust performance and security, but they can be costly and have limited customization options. | Linux servers are known for their stability, security, and flexibility, making them a popular choice for web applications. They are also cost-effective. | Windows servers provide excellent support and integration for .NET applications but can be more expensive and less efficient than Linux for web hosting. | Hosting on mobile devices is not typically viable due to hardware limitations, connectivity issues, and security concerns. |
| **Client Side** | Developing for Mac clients may involve higher costs but offers a premium user experience and robust performance. | Linux clients can be more challenging due to the variety of distributions, but they offer flexibility and are cost-effective. | Windows has a broad user base, making it essential for client-side development, though it can be prone to security vulnerabilities. | Mobile clients require considerations for touch interfaces, screen sizes, and performance optimization. |
| **Development Tools** | Tools like Xcode and Swift are commonly used for Mac software development. | Linux development often utilizes tools like GCC, Python, and Java. | Windows development is typically centered around Visual Studio and .NET frameworks. | Mobile development uses tools like Android Studio, Swift, and cross-platform frameworks like React Native or Flutter. |

## Recommendations

1. **Operating Platform:** After careful consideration, we recommend a cloud-based Linux server for hosting "Draw It or Lose It." This platform strikes an ideal balance between stability, security, and cost-efficiency. Linux's open-source nature allows for extensive customization and scalability, crucial for handling high traffic volumes and diverse user interactions inherent in a web-based game application.
2. **Operating Systems Architectures:** The chosen Linux platform facilitates a microservices architecture. This modern approach to software architecture enables scalable, flexible deployments and maintenance. Microservices allow for rapid updates and feature rollouts, ensuring that "Draw It or Lose It" can adapt quickly to changing user demands and technological advancements.
3. **Storage Management:** For storage, a combination of SQL and NoSQL databases, specifically MySQL and MongoDB, is proposed. This hybrid approach ensures robust data management capabilities, catering to both structured and unstructured data types prevalent in gaming applications. Such a setup is designed to handle the game's complex data requirements efficiently while ensuring scalability and performance.
4. **Memory Management:** Efficient memory management is vital for "Draw It or Lose It," especially due to its need for rapid image rendering. We'll utilize programming languages and frameworks that support effective garbage collection mechanisms, reducing the likelihood of memory leaks and ensuring optimal resource utilization. This approach is geared towards maintaining high performance and responsiveness across all platforms.
5. **Distributed Systems and Networks:** Implementing RESTful APIs is key to ensuring seamless communication across different platforms. This network architecture not only supports scalability but also facilitates the integration of diverse client systems, essential for a multi-platform game like "Draw It or Lose It." We'll address potential network challenges, such as connectivity and latency, with robust error handling and efficient data transmission methods.
6. **Security:** To safeguard user data and ensure secure communications, we will implement SSL/TLS protocols for data transmission and OAuth for user authentication. This security framework will protect against common vulnerabilities and ensure that user information is securely managed, both in storage and in transit, across various platforms.

These recommendations aim to provide "Draw It or Lose It" with a solid foundation for expansion into multiple operating platforms, addressing key aspects such as system architecture, storage and memory management, network integration, and security.